An Analysis of Tabular Reinforcement Learning Algorithms for the Design of Additively Manufactured Acoustic Metamaterials

Undergraduate Research Thesis

Submitted in Partial Fulfillment of the Requirements of Honors Research Distinction to the Department of Mechanical and Aerospace Engineering at The Ohio State University

By

Christopher J. Eubel

The Ohio State University

2020

Thesis Committee:

Dr. David Hoelzle, Advisor

Dr. Manoj Srinivasan

Abstract

In the modern age of advancing technology, led by developments in artificial intelligence, automation, precision manufacturing, and optimized design, it is now commonplace to see revolutionary technologies arise from the fusion of these fields. When advanced technologies can combine their strengths and integrate symbiotically for a worthwhile application, the results are bound to be transformative. In the case of precision manufacturing and optimized design, there would be remarkable benefits of merging the two through means of artificial intelligence and automation technology. An autonomous manufacturing system, capable of understanding the performance of each part it fabricates and being able to self-optimize the design, would undoubtably alter the way certain parts, or even entire systems, are engineered and devised. In this thesis, we present an analysis of several intelligent design algorithms acting on a computational emulation of a manufacturing testbed. This testbed is a closed-loop, fused deposition modeling system for printing acoustic metamaterials designed to achieve a desired acoustic passband. In theory, by using measured performance feedback gained through experimental fabrications, the intelligent system should be able to learn the optimal regions of the design space and find the design parameters to achieve the passband objective. The several intelligent design methods researched and analyzed in this thesis are that of basic tabular reinforcement learning algorithms. Reinforcement learning is thought to be a promising approach because of its potential to be data-efficient and learn online effectively. Through various hyperparameter studies, we hoped to discover strong performing hyperparameter configurations for each algorithm, and then further understand their performance on a simulated environment of the testbed through trajectory visualization. From these studies, relations of the hyperparameters and the importance of balancing exploration with exploitation were discovered.

Acknowledgements

First, I would like to thank the HRL team for welcoming me in and contributing to such a constructive undergraduate research experience. It was great to be a part of such an interesting research project, largely led by Zhi Zhang and Md Ferdous Alam, who both exhibited exceptional attention to detail, a passion for their work, and were always willing to help. I could not have asked for better teammates to work alongside. As for the rest of the members in the lab – Andrej Simeunovic, Ali Abid, and Nathanial Wood – I cannot thank them enough for always taking the time to explain technical concepts, discuss technology, and just chat in the lab. Working with and getting to know such exceptional graduate students was one of the most valuable experiences of my time as an undergraduate. There is so much to gain from being surrounded by those that only encourage you to improve. Lastly, I would like to thank Dr. David Hoelzle for allowing me to be a part of his research team and guiding me over the semesters to help develop my skills as an engineer and researcher. Because of this experience, I cannot recommend undergraduate research enough to those that are interested.

Lastly, I would like to thank the graduate and undergraduate research advisors for helping me along the way, especially Megan Doolin. After learning about the combined degree program, I spent a great amount time conversing with Megan over email and in her office. Without her help and guidance, this experience may have never come to fruition.

Table of Contents

List of Tables

List of Figures

1 Introduction

As artificial intelligence (AI) methods have become increasingly more sophisticated in the 21st century, it is no surprise that engineering design has become a deserving application of this technology. At the highest level, advanced engineering design is typically a repetitive process of synthesis then analysis, by means of engineering fundamentals and computational methods, to generate a satisfactory solution to a given problem. With recent breakthroughs in AI, it is now reasonable to suspect that a data-driven, closed-loop approach can realize comparable solutions for applicable design problems. This may suggest notions of topology optimization, however, this newly proposed method seeks design optimization through the integration of manufacturing and physical experimentation, instead of purely through simulated mediums. It is conceivable that several computational approaches, through various optimization and machine learning techniques, could be used to implement efficient and intelligent design formulation in this manner. The development of such an autonomous framework would be invaluable to industry. Outfitted manufacturing systems would be able to make subtle improvements to well-established solutions, as well as embark into design spaces less understood by the current engineering and science community. The ideal system-framework to implement this novel, autonomous method of engineering design is a manufacturing process that can provide feedback on the fabricated design's performance and easily manipulate its own output. Accordingly, as a testbed for this research, a closed-loop, fused deposition modeling system was developed and utilized to act as the substratum on which an intelligent system – capable of designing an acoustic metamaterials with a specific acoustic objective – could be researched. The goal of working through this specific testbed is that principles and methods for a more general framework of autonomous design will be understood. For this thesis, several tabular reinforcement learning methods for autonomous design will be explored and analyzed based on a simulation of the testbed. In Section 1.1 and 1.2 the testbed and methods of autonomous design will be discussed in further detail, respectively. In Section 1.3 the analysis of several reinforcement learning algorithms, the topic of this thesis, will be introduced.

1.1 Phononic Crystal and Additive Manufacturing System

To facilitate this research, a physical testbed was devised and constructed to represent a suitable manufacturing framework in which an autonomous design system could be implemented. This generic framework can fabricate a part for a specific design objective, measure the part's performance against that objective, and then, using that feedback, autonomously update the design's mutable parameters to converge to an optimal solution. This framework suites the ease of development, implementation, and analysis of artificially intelligent design methods, which is the main ambition of this research and thesis. In our case, the design objective at hand is to design and 3D-print a cuboid, lattice-structure-based acoustic metamaterial – more appropriately named a phononic crystal (PC) – that has a specifically desired acoustic passband spectrum. Meaning, when a broad, high-frequency vibrational wave is incident on the PC, only certain acoustic frequencies will pass through and the rest will be mainly absorbed. An achievable acoustic passband is the quantitatively described design objective and the intelligent system is granted two mutable design parameters to attain it. These parameters are the filament diameter, d, and lattice spacing, l_{xy}, of the PC's structure. Details of this can be seen in figure 1.1, below. This design problem was specifically chosen as the testbed because these specific acoustic metamaterials are simple to design, fabricate, and analyze. Also, significant research on these PCs by a former lab member, Chaitanya Vallabh, had previously been published [1], so their utilization did not require intensive prior research to understand the acoustic properties and fabrication methods.

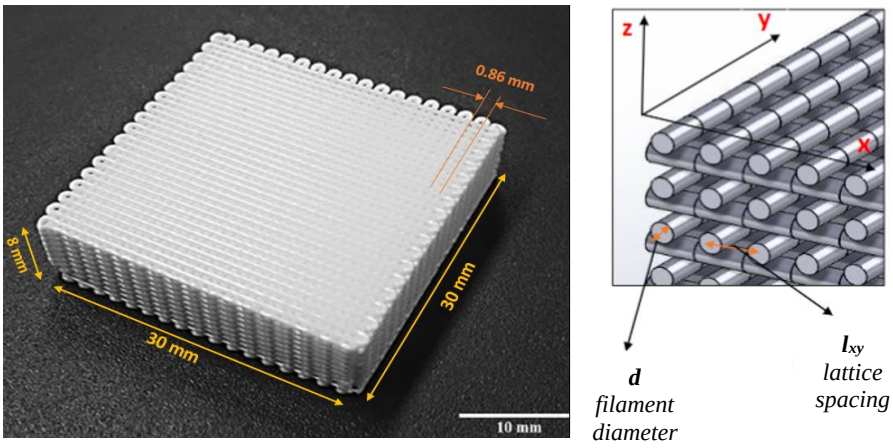

Figure 1.1: (left) example of phononic crystal and its scale (right) design parameters of the phononic crystals structure

Since the PC is FDM-printed, the lattice structure is fabricated one layer at a time with the parallel rows of the lattice being rotated 90º every layer. Unlike in the right image, the fabricated PC have continuously connected layers as seen in the left image [1]. For adhesion purposes, the first layer is flattened to the bed – unlike the rest – but this likely does not impede its acoustic properties as a metamaterial develops its properties from periodicity. If it does impact the properties, it will be irrelevant to the RL algorithm's ability to learn and compensate for this nonuniformity.

The fabrication system used was a modified Ultimaker 3D Printer and can be seen in figure 1.2, below. This printer was modified to have a rotating print bed, acoustic transducers, and automation software. The rotating print bed allows the system to print a single PC, rotate it under the acoustic transducers for testing, and then rotate to the next location to print another PC. This cycle can run continuously for twelve PC samples without human intervention. Human intervention is only needed to remove the samples once the bed is full. Acoustic transducers actuated by solenoids, placed above and below the bed, can clamp down to measure the sample's acoustic properties to provide the feedback on the PC's passband properties. With this, the automation software can process the feedback data, produce a new design, generate the associated g-code, and begin the print-test cycle again. It should be noted that additive manufacturing is highly useful for this research and proposed manufacturing framework because of its flexible ability to fabricate a design purely from mutable code.

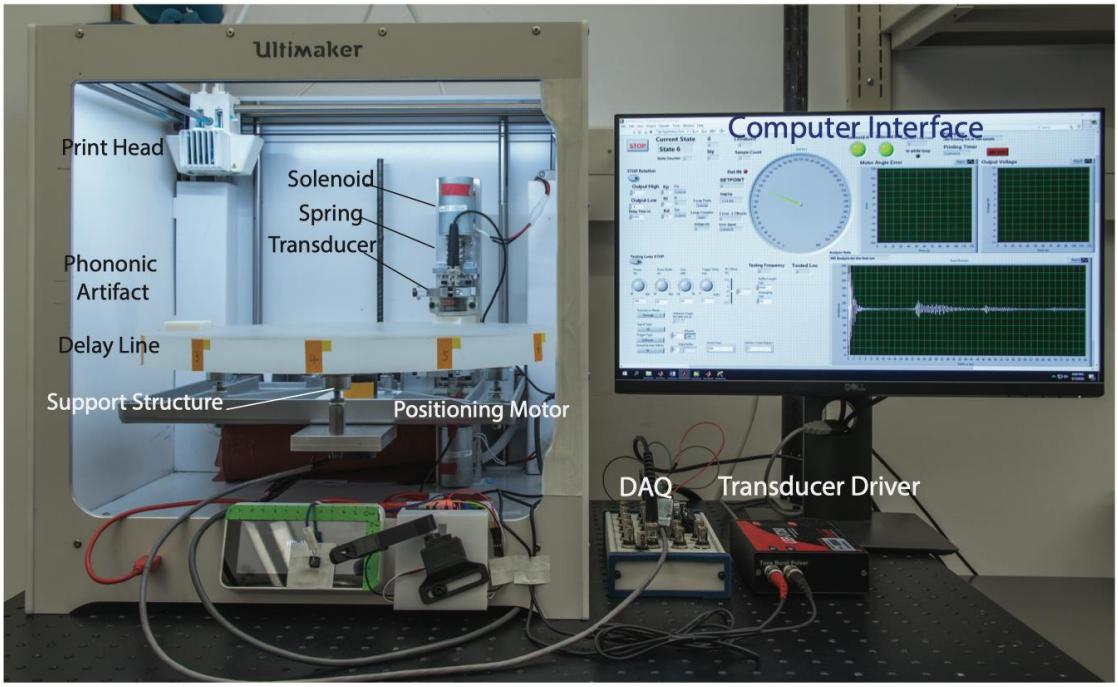

Figure 1.1: modified Ultimaker printer system used for physical testbed

All physical components are visible and labeled in this figure. The rotating bed was carefully constructed from rexolite so as to not impede the vibrational signal and precision of its measurement. The automation software consists of many working programs all commanded logically by a LabVIEW backend. The computer monitor displays the LabVIEW frontend which is a user-interface for controlling and diagnosing the printer system. Various MATLAB and Python programs were integral in the design and fabrication of the PCs. Ultimaker Cura was used to transmit the pre-sliced PC g-code to the printer.

1.1 Reinforcement Learning and Tabular Methods

The most important element of our testbed is the method of autonomous design and its implementation. This element performs its role between the testing and printing stage of the PCs fabrication. From the processed performance feedback of the previously printed PC, the system must determine changes to the design parameters to move closer towards the passband objective. Please reference figure 1.3 below and figure X in the Appendix for the process flow of the testbed system.

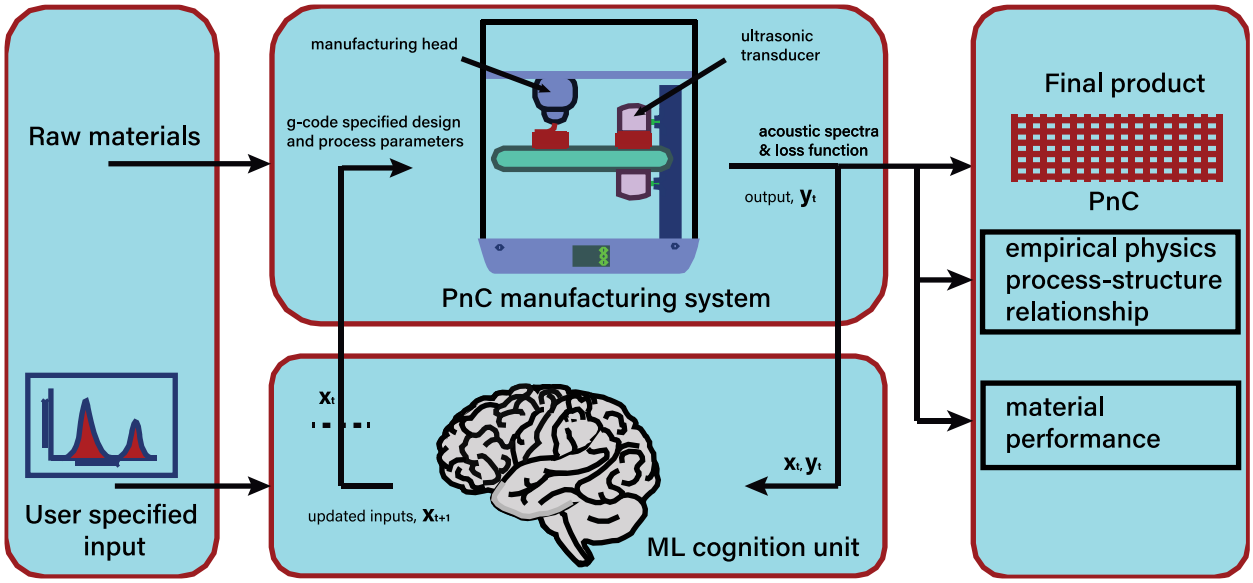

Figure 1.2: general autonomous manufacturing process for the PC testbed process

First, the PLA (raw materials) and user-specified acoustic passband (via the reward function) are inputted to the system. Then the printer will fabricate and test a part for its design performance. If the part is not of acceptable performance, it will be discarded, and its data will be fed back to the system for learning and an opportunity to update the design. The print, test, and redesign process will iterate until a satisfactory PC is designed. From this point, the system can choose to manufacture the part as needed with the fixed design or continue optimizing and manufacturing a satisfactory part simultaneously. The machine learning (ML) cognition unit represents the RL algorithm in use.

The methods used in this thesis are within the class of reinforcement learning (RL) algorithms. RL is one of three main fields of machine learning; the others being supervised and unsupervised learning. Without going into too much detail, the general RL framework is unique because it is based around interaction with an environment and learning from data generated by those interactions, a process analogous to the rudimentary learning done by animals and humans. At a general level, this interaction with the environment is commanded by a software agent that acts on the environment, then in response, the environment presents an updated state and returns a calculated numerical reward to the agent. This action-state-reward sequence repeats, creating a trajectory across the state space as the agent attempts to understand the environment it is in and maximize the long-term reward acquired. RL has a wide variety of applications and associated algorithmic implementations, but considering the somewhat low dimensional problem at hand, tabular algorithms were utilized: Q-learning, double Q-learning, SARSA, and expected

SARSA. This omits neural network based RL algorithms. Tabular methods are categorized as such by having a discrete container for each position, and therefore, are only effective when the space is sufficiently small, thus avoiding the curse of dimensionality. The RL framework will be discussed in more detail in the Background section.

1.2 Algorithmic Analysis with Simulated Environment

Reinforcement learning algorithms have several mutable parameters, called hyperparameters, influencing the process of the agent's learning and decision-making that affects the overall behavior of the algorithm, and consequently, its performance within a particular application. Hyperparameters are manually set and are distinct from the parameters that help construct the machine learning model. The hyperparameters analyzed in this thesis are the learning rate (α), discount factor (γ), and epsilon (ε) which are variables for numeric values between 0 and 1. Roughly speaking, the learning rate determines how drastically the agent adjust its perception about a particular position within the environment, the discount factor determines how much the agent values future reward, and epsilon pertains to ε-greedy method which determines how the agents selects its actions. To understand how the values of these hyperparameters affect the performance of the selected algorithms, within the application of our testbed, hyperparameter studies were performed on a simulated environment. This simulated environment, based on FEM simulation on the acoustics of the PC structure, is a virtual and interactive embodiment of the actions, states, and approximate rewards present on the physical system. Alongside this study, the more important question of whether or not these algorithms are able to accomplish the design objective will be analyzed and discussed. The exact implementation and purpose of these parameters and the details of the simulated environment will become clearer in the next section.

2 Background

2.1 Testbed Environment

The environment for our testbed consists of the design-parameter state, design-change actions, and reward for the acoustic performance. Again, the design-parameter state, or just state, is the two mutable parameters of the PC design – the filament diameter and lattice spacing. The filament diameter, d, has been discretized from 300 µm to 635 µm with 5 µm spacing and the lattice spacing, $l_{xy,}$ has been discretized from 700 µm to 1035 µm with 5 µm spacing, resulting in 68 potential choices for each parameter. It should be pointed out that the physical details of the state, or design parameters, are meaningless to the agent and its ability to learn.

The design-change actions, or just actions, are how the software agent can choose to adjust the state. The agent has eight actions in which it can adjust its design parameters towards contiguous states. Because the environment is two-dimensional, one can imagine the set of design states as a 2D cartesian grid and the actions as the agent's ability to change its position on the grid to any discrete state adjacent to the current position. Of course, actions become constrained when at the boundaries of the parameter limits. A representation of a non-constrained agent action-selection scenario is shown below in figure 2.1; this type of visualization becomes imperceivable as the number of state dimensions increase or if the state or action space is not discretized.

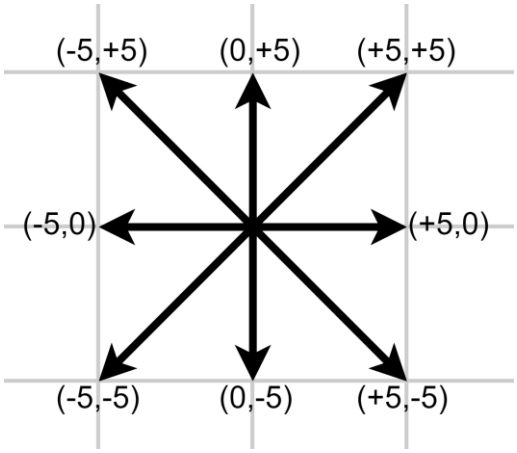

Figure 2.1: visualization of agent's action selection options within the state space of the environment

The agent has 8 options on how to change the PC's design state. This is shown visually with the change of coordinates, shown in the image, that have the unit of μm. Many low-dimensional RL problems are posed and visualized in a similar way to facilitate the education process. For the sake of this research, visualizing a lower dimensional process is useful to understand and evaluate the performance before increasing the amount of design variables of the agent and state.

Lastly, we have the reward which is used to quantify a particular design's ability to achieve the desired acoustic passband, thus giving the agent an assessment of how well it is performing at a state. This reward is calculated by quantifying key differences between the desired and measured acoustic passband spectrum. On the physical system, this is performed from the feedback acquired by the acoustic transducers. A method for computing the performance loss can be seen below in figure 2.2. The reward is equal to the calculated loss subtracted from an arbitrary positive bias such that all reward values are positive. This operation inverts the relative relationship of the loss values – high loss becomes low reward and low loss becomes high reward.

Loss function:

$$\mathscr{L}(y_d, y_a) = \mathscr{L}_1 + a\mathscr{L}_2 + b\mathscr{L}_3$$

where,

$$\mathscr{L}_1 = \sum_{i=1}^{m} |y_{ai} - y_{di}|$$

$$\mathscr{L}_2 = \frac{|f_{ac1} - f_{dc1}|}{f_{dc1}} + \frac{|f_{ac2} - f_{dc2}|}{f_{dc2}} + \frac{|f_{ac1} - f_{ac2}|}{|f_{dc1} - f_{dc2}|}$$

$$\mathscr{L}_3 = \frac{|y_{ac1} - y_{dc1}|}{y_{dc1}} + \frac{|y_{ac2} - y_{dc2}|}{y_{dc2}}$$

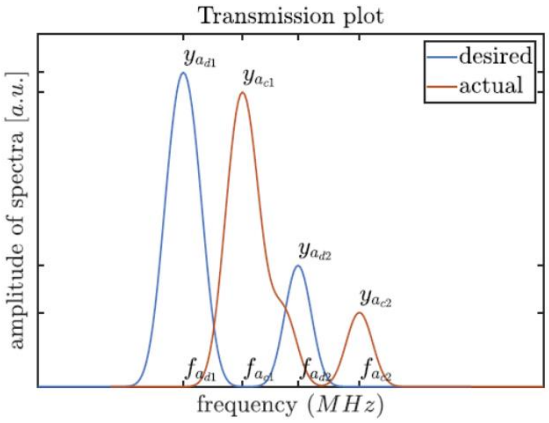

Figure 2.2: loss calculation between the desired passband spectrum and actual passband spectrum

This loss function is a fundamental component of the reward signal. Without it, there would be no way to quantify the design performance and, thus, the agent would have no means of improving. Notice that absolute values are utilized such that the total difference of all spectrum elements are considered. Also, this dual passband spectrum is generally representative of what an arbitrary PC's passband spectrum looks like.

Not only can this action-state-reward framework be realized on the physical testbed, it can also be modeled on a simulated environment through a digital emulation. This emulation is a virtual and interactive representation of the actions, states, and approximate rewards present on the physical system. The greatest challenge in developing a digital emulation of the physical testbed, however, is generating the reward which requires an FEM simulation of the passband spectrum for a given design state. If this can be done accurately enough, though, there is little difference between the real environment and simulated environment. It should be noted that for the simulated environment, the generated reward at a given state is constant for all time steps and therefore the computational emulation is a deterministic environment, rather than a stochastic one. This inherent detail is important to remember as it unalike to the physical system and will impact the results of the hyperparameter studies and performance of algorithms.

2.2 Reinforcement Learning and Markov Decision Process

"Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal" [2, p. 1]. This section will expand on the fundamentals of

reinforcement learning and the Markov decision process. The general ideas have been introduced above, but proper definitions and a more formal exposition will be presented below. However, this will not be comprehensive; discussion will be limited to what is relevant to this thesis for the sake of clarity and conciseness. The technical topics below are organized by a general elaboration followed by their relation to the application of this research. This content is summarized from *An Introduction to Reinforcement Learning* [2].

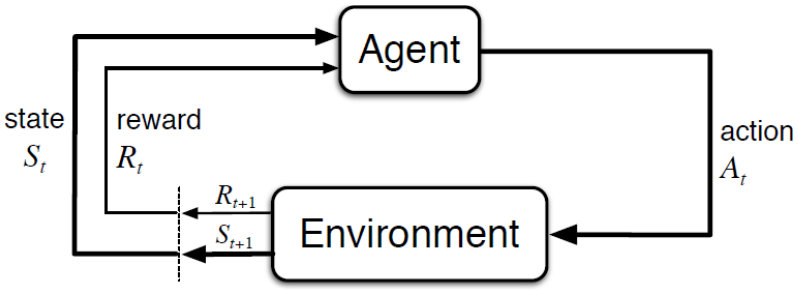

Figure 2.3: The standard agent-environment interaction process of a Markov decision process

The Markov decision process is usually characterized by a 5-element tuple (S,A,P,R,γ). S is a finite set of states, A is a finite set of actions, P is a transition-probability matrix, R is a reward function, and γ is the discount factor. These 5 elements can formulate a mathematical framework of sequential interaction between and agent and an environment. An agent will act on the environment, and in return the environment will return a new state and reward. The transition-probability describes the probabilistic relations between states and selected actions, and the reward function determines what numerical reward is returned [2, p. 48].

2.2.1 Markov Decision Process and State, Action, Reward

A Markov decision process (MDP) is a mathematical formulation for the process of sequential decision making [2, p. 47]. Figure 2.3, seen above, is a diagram of this process. An MDP helps to systematically establish the agent-environment interaction for a certain scenario. Starting with the agent, it can act on the environment with an action A_t from the set of available actions , then in response, the environment will feed back an updated state S_{t+1} and numerical reward R_{t+1} from the set of available states and rewards. The time-subscripts of the state, action, and reward are only to clarify the sequential nature of the process. The new state and reward become the current state and reward, and the process repeats

itself at the next time step. It should be noted that MDPs exhibit the Markov property, which means that all information needed to understand the transition from the current state and action to the following state and reward is contained within the current state. That is, all necessary memory to describe the transitions must be incapsulated within the environment's present state. These transitions from time step to time step are equated through defined transition probabilities. For our testbed, because the action taken results in a consistent and known result of the future state, we say the environment has a deterministic state transition whose probabilities are trivial. The standard state-transition probability equation can be seen below.

$$p(s'|s,a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a)$$

This general process is a robust framework that can be applied to many applications. Accordingly, this formulation can be used to frame a reinforcement learning problem, as with the case of this research. As discussed in earlier sections, the state is the PC's design parameters, the agent's action is how the design parameters are changed, and the reward is a quantification of the design's ability to match the desired acoustic passband. But, even with all this established, it is still reasonable to wonder how an autonomous agent goes about achieving the task intended. This is where reward plays its essential role. Based on the idea of the reward hypothesis, a goal or purpose can be communicated through the maximization of the expected return of cumulative reward [2, p. 53]. Thus, with a carefully crafted reward function, it is convincing that an agent could learn and carry out any task desired that is within the means of the agent's actions and surrounding environment.

2.2.2 Policy π

Generally speaking, a policy – notated as π – dictates how an agent behaves – how it learns and how it acts. Typically, the agent has two policies: the behavior policy and target policy. The behavioral policy is how the agent goes about selecting the actions to take and the target policy impacts how the agent updates its understanding about the states and actions within the environment [2, p. 103]. The target policy will make more sense in the next sections after understanding the value function and temporal-

difference learning. The distinction of these two policies is important because it grants flexibility to the agent and establishes off-policy and on-policy methods. Off-policy is when the behavior and target policy are different, and on-policy is when they are the same. Both methods appear in the algorithms of this thesis, and their distinction and usefulness will be outlined in Section 3.2.

The behavior policy addresses one of the great quandaries of RL – exploration versus exploitation. This problem describes the agent's challenge of striking the proper balance between maximizing reward and expanding its knowledge about the true nature of the environment. The agent must seek out the greatest reward in order to pursue the desired objective, but it must also improve its beliefs about the environment to make thoughtful decisions. There are a number of behavior policies that address this trade-off, but the one utilized in this thesis is the simple ε-greedy method. Greedy action selection is when the agent selects the action that will result in the greatest expected benefits. The ε-greedy method is described as near-greedy because with a probability of ε a random action is chosen, thus exploration, and with a probability of 1-ε the maximal action is selected, thus exploitation. Remember that ε is a numeric value between 0 and 1. In the case when ε is 1, the ε-greedy method becomes purely greedy action selection.

2.2.3 Value Function

In general, value is defined as the sum of potential future rewards. This quantity can be attached to a distinct state, as in state-value, or a distinct state-action pair, as in action-value. The state-value is used to quantify the benefit of being at a given state, and likewise, the action-value is used to quantify the benefit of taking a particular action at a given state. In either case, the agent's goal is to estimate, or learn, the value function – a mapping of states or state-action pairs to their value – to make the most intelligent action selection based off those judgements [3]. For tabular algorithms, the numeric value for every discrete state or state-action pair of the environment represents the value function and is called the Q-table. Furthermore, even though we have immediate reward available at every state, value is more useful because it allows the agent to compile information on long-term reward, and maximizing the cumulative

reward in the long run is the agent's innate goal. Surprisingly, a state with the highest reward is not always the best state to be in and value estimation can help decode this enigma. For instance, a state with high reward, only reachable through a path of negative rewards, will reflect a lower value as the consequences to and from that state are not worth its reward. Finally, the process of learning the value function is not immediate, but an iterative process through continual trials of agent and environment interaction.

The notion of cumulative reward is defined by the return, G_t, which is a sequence of rewards accumulated over a specific amount of time.

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

In cases where the time span is infinite, that is T = ∞, discounting this return is necessary to produce a finite value. Discounting the subsequent reward operands results in discounted return, which is dictated by the discount rate, γ – a hyperparameter with a value between 0 and 1. In essence, the discount rate reflects how much the agent accounts for future reward and the appropriate value for the discount rate depends on characteristics of the application. More often than not, reward sooner is preferred to reward later which is reflected by a discount rate less than one.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Now with this new term, we can say the agent's ultimate, long-term goal is to maximize return, and we can revisit the discussion of the value function. Because value is based on the future reward gathered it must also be based off the agent's future actions, or policy. Therefore, when talking about a value function it must always be referenced to a specific policy, π. The state-value function for policy, $v_\pi(s)$, is defined as the expected return starting at a state and following π.

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S}$$

Taking this a step further and including taking an action, we have the action-value function, $q_\pi(s, a)$.

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

Nearly all reinforcement learning algorithms attempt to learn one of these value functions. But, it should be noted that in most cases the agent never actually learns the true values as this can be impossible or require infinitely many visits to each state. Although, estimating the approximate value is nearly as good for acceptable performance.

2.2.4 The Bellman Equation and Temporal-Difference Learning

The Bellman equation, originating from dynamic programming, defines the expected value of a particular state for a given policy. Value is defined by a recursive relationship because the value of one state is equal to the instantaneous reward plus the value of the next state, and then the value of the next state is calculated in the same manner, and so forth. And when equating expected value, all next states and their probabilistic values are incorporated which results in an exploding tree of branching recursion. The expansion of the value function for a policy is shown below, with the Bellman equation defined by the last equation. From this, it is evident that the Bellman equation is simply a system of nonlinear equations, with one equation for every state [2, p.64]. Furthermore, we know the Bellman equation must hold for the value function at each state. Therefore, a value function is simply verifiable by computing the Bellman equation at all states and verifying equivalent results of the value. However, this is not of much use as the value function is learned, not given, and the Bellman equation is largely unfeasible to solve.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

Temporal-difference (TD) learning is a fundamental method of RL that is driven by differences in successive estimations of the same quantity [2, p. 20]. In the case of the TD-based tabular algorithms of this thesis, this quantity is action-value. TD learning combines the two original paths of RL, differentiated by methods trial-and-error and value functions, into one method [2, p. 35]. Likewise, TD learning combines ideas from Monte Carlo methods and dynamic programming. Before embarking onto the TD concepts, it is important to understand the ideas of the prediction problem and control problem within RL. The prediction problem is the challenge of estimating the value function for a given target policy and the control problem is about finding the optimal policy [2, p. 119]. The simplest TD method, one-step TD, uses the update rule below to solve the prediction problem.

$$
V(S_t) \leftarrow V(S_t) + \alpha \Big[\underline{R_{t+1} + \gamma V(S_{t+1})} - V(S_t) \Big]
$$

An update is how the agent amends its understanding about a particular state or action value. The underlined portion is considered the sampled value estimate, or target, and the current value estimate is what is being subtracted from it to quantify the difference between the agent's current value belief with the newly sampled value. This difference operation, defined as the TD error, is performed and added to the current value estimation at each time interval. In other words, learning is happening for every step made by the agent, a distinct advantageous feature of TD learning. The variable multiplying the TD error is the learning rate hyperparameter, α. This variable determines how drastically the TD error impacts the result of the new value estimate over the old value estimate. Again, α is a value from 0 to 1 with higher values causing a more impactful update. Lastly, TD learning is considered a bootstrapping method because this update to the value estimation is based on an existing value estimate, and not the true value.

The four tabular algorithms in this thesis utilize a form of the TD update rule to estimate the action-value function. The TD update equation can be seen more generally in the equation below, and the element unique to each algorithm is how the target is defined and calculated. The exact implementation of this rule will be discussed in the next section for each algorithm.

$$NewEstimate \leftarrow OldEstimate \; + \; StepSize \left[Target - OldEstimate \right]$$

2.2.5 Offline and Online Learning

Offline and online learning are distinctions of where the machine learning model is learning and operating or, alike, where the input data is coming from. For RL, due to the data being attached to the environment and its lack of parallelization, the distinction between the two is less apparent, but it is still useful to understand. If the model is learning from a static batch of training data, then this is offline learning. If the model is learning from sequentially incoming data from the application, then this is online learning. In general, machine learning models will usually be trained offline from a large database, then they will be deployed to their application via transfer learning where the model can either remain static or continue learning in an online manner. The latter has advantages because it allows for continual improvement and adaption to the real-world application, but requires more work to implement and maintain. In the case of the application of this research, the design framework would likely have to learn online as each part is fabricated, unless the application scenario has already compiled a database from many produced parts or simulation. For this thesis, learning is done in an offline fashion from static simulation data, but this will still give insight on the algorithm's ability in an online scenario.

2.2.6 Continuing Task

Reinforcement learning problems can be organized into two classes that differentiate the nature of the task. First, there are episodic tasks which break up the sequence of interactions into episodes. Each episode ends in a terminal state after which the agent will be reinitialized and begin interacting again in the next episode. A common example of an episodic task is the game of chess, where each game equals

one episode. The agent can learn through multiple episodes, but its goal is still to maximize return over a single episode. On the other hand, there are continuing tasks which are not broken up into finite-length sequences. The agent is to learn and try to maximize return continuously without a terminal state. This is the presumed task for the manufacturing application of the intelligent design framework. Throughout the intelligent system's life, it should continuously be trying to create the globally optimal part and adapt to an any subtle changes to its environment.

2.3 Why Reinforcement Learning?

Among a few possible methods that could be used to power an intelligent design mechanism, RL was chosen for its potential to reduce the amount of data required and its ability to effectively learn online. With many other methods, effectiveness requires sampling much or all of the space to generate models of the environment. In our case, we are not concerned with areas of the environment with low reward and therefore should avoid wasting energy on collecting and learning that data. Ideally, the agent's trajectory across the environment should be minimal, traversing only where is needed to reach an optimal or near-optimal solution. Recalling the intended manufacturing application of this framework, it would become awfully expensive in time and capital to understand the entire design space, especially as the number of dimensions increase. In most scenarios, as with this research, a completed part would equate to a single data point of reward. Also, unlike this research, most real-world applications would not have a simulated environment; meaning efficient, online methods would be necessary for the practical application of this technology. A simulated environment could be developed for the specific application, but then the necessity for efficient design algorithms is being traded for the need of a costly or likely impossible simulation software. The gravity placed upon the rate of convergence, or efficiency, of the design algorithms is worth acknowledging but it is outside the scope of this thesis.

3 Methods

3.1 Simulated Testbed Environment

The testbed's simulated environment is essentially an interactive software emulation of an MDP for the testbed application. The agent, always beginning at a random state, can select an action and take a step to receive the corresponding new state and reward. This process repeats as the agent attempts to learn the value function and maximize reward. Again, the reward of a state is equal to its numerical loss subtracted from a positive bias of 300. Because the environment and state transitions are deterministic, there is no ambiguity in the resultant next state nor inconsistency in the reward obtained. In application, the agent will be performing a continuing task on the environment but for the sake of performing an analysis on the algorithms, the need for episodes emerged in order to equate average performance. However, this does not mean it is an episodic task. For all episodes of the studies the agent is seeking to reach the same objective. That is, the desired acoustic passband spectrum, and consequently the reward surface, never changes. This is acceptable because at the start of each episode the agent's memory of the environment is wiped clean. An example of an agent's trajectory across the reward surface can be seen in figure 3.1, below. The agent starts from a random state and its trajectory ends after a fixed number of steps. This figure is representative of what is performed thousands of times throughout the performance studies of this thesis. Note the discrete states and somewhat-noisy, discrete reward distribution of the reward surface.

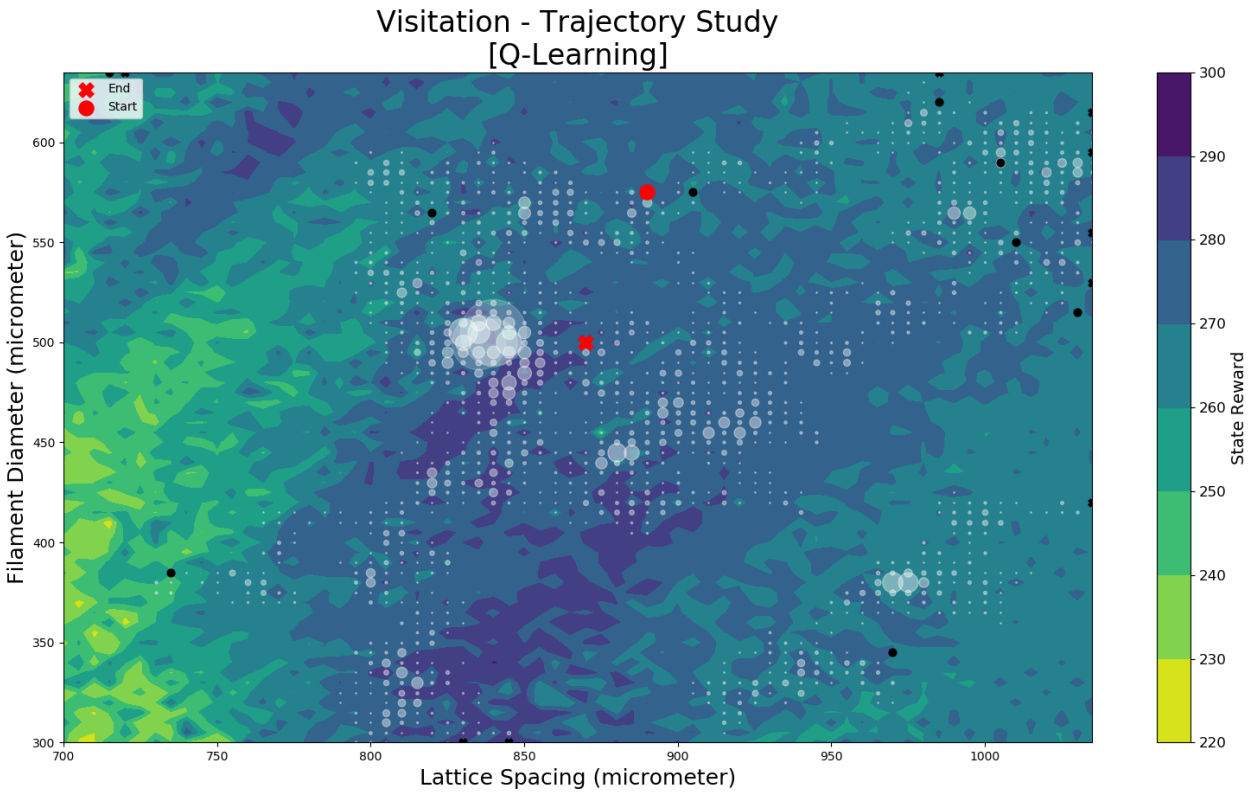

Figure 3.1: example of agent trajectory and visitation rate on simulated reward surface

This trajectory plot shows the reward surface superimposed over the state space of the PC design. The white, transparent markers signify the states the agent has visited, with their size representing the total number of visits. The black dots signify the randomly initialized state of the agent after it had stepped outside the boundary of the state space.

It should be pointed out that there are slight differences between the hyperparameter and performance studies with regards to the environment and greedy action selection. For the hyperparameter studies, when the agent is at the edge of the environment's boundary, any actions taken towards nonexistent design states will result in a reward of zero and the agent will remain in the original state. Also, greedy action selections at a state with multiple equally maximal actions-values will result in the first occurring maximal action in the action-space array. This is a result of the software selecting the first applicable value from a list. Since unexplored states have all action-values equal to 0, an action will always result in a rightward action, which translates to +5 μm of l_{xy} and +0 μm of d. For the performance studies, any action taken outside the state boundaries will result in a random reinitialization onto the state space and a reward of 0. For greedy action selection with multiple equally maximal action-values to

choose from, an action from that maximal subset will be randomly chosen. The implications of these differences will be revisited in the Results and Conclusion sections.

3.2 Tabular Algorithms

The four algorithms used in this thesis are in the class of tabular methods. A tabular approach is sufficient when the state and action space is appropriately small with respect to the means of the application. If the space is too large, this approach will fail due to time complexity and memory issues brought upon by the curse of dimensionality [3, p. 27]. This is due each state or state-action pair requiring a discrete container for the estimation of its value. From all these containers, one can abstract an array, or matrix, to organize and lookup the agent's value-perception of the discrete environment – this is where a Q-table emerges. In the simplest of cases, the Q-table can be visually superimposed over the discrete, grid-like environment with partitions for the action-values. More standardly though, the Q-table organizes action-value estimation into a table where the rows represent each possible state and the columns indicate the actions. An example of a simple Q-table can be seen in table 3.1, below. In the case of this thesis, the Q-table is a three-dimensional array. The two design parameters create a two-dimensional grid of all possible states, and then each layer in the third dimension consists of one of the eight actions available to the agent. Although being able to visualize the Q-table can be helpful, the exact details of its structure are usually only relevant to its compatibility with the logic of the software.

Table 3.1: example of Q-Table with an arbitrary number of states and four actions

	Actions			
States	Up	Down	Left	Right
State 1	1.143	1.657	0.179	1.726
State 2	0.989	0.308	-0.917	1.656
State 3	-1.001	0.474	-1.496	0.426
State 4	1.814	2.309	0.493	-1.222
State 5	1.771	-1.493	-0.938	-1.179
State 6	1.368	0.118	1.472	-1.394
…	…	…	…	…

The Q-table is a tabular representation of the estimated action-value function. Assuming the agent is choosing the greedy action, in State 4 the agent will select the Down action. The Q-table does not store any state-transition or probabilistic information, only the perceived value of selecting a certain action at a specific state. A Q-table is very different from neural deep Q-learning methods that use neural networks instead to approximate the action-value function.

For understanding the tabular algorithms studied, it is important to recognize their similarities and differences. For the similarities: each algorithm is a form of TD estimation, ε-greedy action selection is utilized for at least the behavior policy, they attempt to learn the action-value function, they are model-free, the hyperparameters are consistent, and all Q-tables are initialized to zero. As for the general process of the algorithms: they all begin by initializing the first state and the action-value function (Q-table), then there is a repetitive interaction with the environment, with each step followed by an update to the action-value function. This process will continue until stopped. The algorithms and their distinctions will be outlined below. Finally, remember that this application is not an episodic task and the outer episode-loop and terminal state in the pseudocodes, figure 8-11 below, should be disregarded. Again, the notion of episodes, or trials, only emerges for the purpose of the studies.

3.2.1 Q-learning

Q-learning is a basic off-policy TD control method. Its pseudocode is in figure 3.2, below. Its pseudocode is in figure 8, below. Being off-policy, it can estimate the action-value function of a policy independent from the behavior policy dictating the agent's actions. Thus, notice that the target in the TD

update uses the maximum Q-value available at the next state instead of the Q-value from the next selected action and state. This separation of policies can be useful because the agent can explore stochastically via the behavior policy, whilst still learning the action-value function of the deterministic and greedy, thus optimal, target policy [4]. This learned action-value function is called Q, equated by the Q-table, and is an estimate of the optimal action-value function. Moreover, the agent is learning from data it would have otherwise not acquired if it were only following the target policy. In short, the agent can explore and learn the optimal actions simultaneously resulting in a more general and powerful method. This does come with drawbacks, however. Off-policy methods often have greater variance and slower convergence than their on-policy counterparts. [2, p. 103]

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Figure 3.2: Q-learning algorithm pseudocode [2]

3.2.2 Double Q-learning

Double Q-learning follows the same off-policy process as the Q-learning algorithm except for the use of two Q-tables, as implied appropriately by its name. The pseudocode can be seen in figure 3.3. The greedy action is selected from the combined action-value functions of the Q-tables and then target updates are made only to one Q-table with either one being selected only 50% of the time. Also, the TD update uses the greedy target policy, as with Q-learning. The purpose of this process is to help combat maximization bias. Because the greedy of the target policy selects the maximal valued state-action pair to

update the estimates, a positive bias of the action-value function can arise. This method of combating positive bias with two Q-tables is called double learning and this idea can be applied to other algorithms [2, p. 136].

Double Q-learning, for estimating $Q_1 \approx Q_2 \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q_1(s, a)$ and $Q_2(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, such that $Q(terminal, \cdot) = 0$
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using the policy ε-greedy in $Q_1 + Q_2$
    Take action A, observe R, S'
    With 0.5 probabilility:
      $Q_1(S, A) \leftarrow Q_1(S, A) + \alpha\Big(R + \gamma Q_2\big(S', \operatorname{argmax}_a Q_1(S', a)\big) - Q_1(S, A)\Big)$
    else:
      $Q_2(S, A) \leftarrow Q_2(S, A) + \alpha\Big(R + \gamma Q_1\big(S', \operatorname{argmax}_a Q_2(S', a)\big) - Q_2(S, A)\Big)$
    $S \leftarrow S'$
  until S is terminal

Figure 3.3: Double Q-learning algorithm pseudocode [2]

3.2.3 SARSA

SARSA, an acronym for state-action-reward-state action, is an on-policy TD control method. So, this method attempts to estimate the action-value function for the behavior policy defined by ε-greedy. This method is different from Q-learning because it operates on successive state-action pairs dictated by the behavior policy, however, this is not unique as this is how an on-policy method must function. As seen in the pseudocode in figure 3.4, the TD update requires the action-value of the current and previous state-action pairs. This on-policy update also guarantees that SARSA will never converge to the optimal policy or learn the true action-value function if there is stochasticity in the behavior policy [4]. However, this can be remedied by modifying ε in ε-greedy method to equal 1/t as each state-action pair is visited an infinite amount of times [2, p. 129].

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Figure 3.4: SARSA algorithm pseudocode [2]

3.2.4 Expected SARSA

Expected SARSA is a variant of SARSA, but really, it follows the Q-learning process more closely. The pseudocode for this algorithm is that of the Q-learning, in figure X above, except for the TD update of the action-value function. Instead of using the maximum Q-value of the next state for the target update, as in Q-learning, Expected SARSA uses the expectation of the Q-value at the next state. Expectation of the Q-value is the sum of the probability of each action being selected multiplied by its Q-value. This more complex methods is advantageous because it overcomes the on-policy variance of the Q-value update, as in SARSA, due to the occasionally random action selection [2, p. 133]. Plus, this advantage does not impede on its guarantees of convergence [4].

3.3 Studies and Analysis

Two forms of computational studies and corresponding analysis were performed with the algorithms above via the simulated environment. A hyperparameter study was done to understand the comparative performance of a wide range and combination of hyperparameter values for optimal selection. Then an algorithmic performance study, built on top of the identified favorable hyperparameters, was done to examine how well the algorithms suited the objective of intelligent design

on the simulated testbed. Various methods of data collection, analysis, and visualization were utilized. The general process performed for each algorithm is outlined in figure 3.5, below.

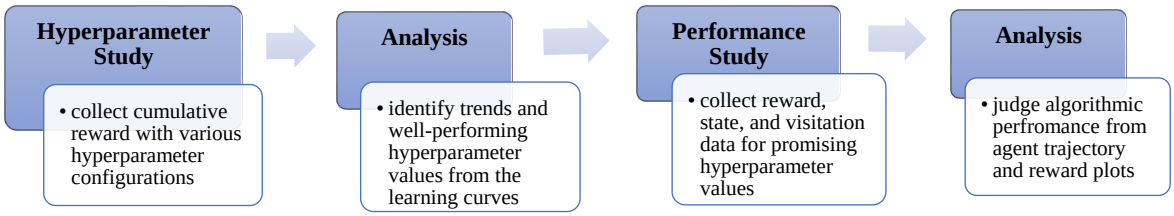

Figure 3.5: summary of process for the Hyperparameter and Performance studies and their analysis

3.3.1 Study Details

The hyperparameters from the TD-update and ε-greedy action selection were analyzed in an iterative and consistent fashion for each algorithm. First, α was varied from 0.01 to 1 at 40 evenly spaced intervals with γ equal to 0.95 and ε varied from 0.025 to 0.2 at intervals of 0.025. This will be referred to as the Alpha Study. Then, γ was varied from 0.01 to 1 at 40 evenly spaced intervals with α equal to 0.5 and ε varied in the same manner as before, and this will be referred to as the Gamma Study. Reward data was collected each step for these two 2-dimensional studies. The exact specification of the α and γ values is irrelevant as the small interval spacing provides enough resolution to understand the trends. Although ε is an independent variable in these two studies, less values were studied because ε-greedy is one of several action selection methods and is not inherent to the tabular algorithms. The range of ε values selected provided a generally accepted balance between exploration and exploitation. For every combination of values for these two studies, the agent-interaction experiment lasted for 10,000 steps and was repeated over 10 trials. Repetition was done to calculate average data. As the agent interacted with the environment it stored cumulative reward at 1000-step intervals for the analysis explained in the next section. This non-continuous tracking of reward was done to save memory and writing time. In summary,

for every parameter configuration and step interval, cumulative reward was collected and average for 10 results. The hyperparameter study details are outlined in table 3.2, below.

Table 3.2: hyperparameter values and study details

Hyperparameter Study Specifications						
	Hyperparameter Details			**Study Details**		
	TD Update		E-greedy			
Algorithm	*Alpha*	*Gamma*	*Epsilon*	*Steps*	*Trials*	*Data Acquired*
Q-Learning	0.1:1:40	0.1:1:40	0.025:2:8	10,000	10	Cumulative Reward
Double Q-Learning	0.1:1:40	0.1:1:40	0.025:2:8	10,000	10	Cumulative Reward
SARSA	0.1:1:40	0.1:1:40	0.025:2:8	10,000	10	Cumulative Reward
Expected SARSA	0.1:1:40	0.1:1:40	0.025:2:8	10,000	10	Cumulative Reward

The hyperparameter values are described by X:Y:Z. X is the first value, Y is the inclusive ending value, and Z is the number of linearly spaced values between the bounds.

The next study was focused on evaluating each algorithm's performance on the simulated environment. For this application, performance describes the agent's ability to discover and reap high reward areas of the environment. A single configuration of hyperparameter values per algorithm were used and were selected from the promising values discovered in the hyperparameter study. Because a three-dimensional study of the hyperparameters was not performed, the most favorable values had to be approximately selected by analyzing and comparing noisy results from the Alpha and Gamma Studies. If multiple values look favorable, the mean value was selected. For each set of hyperparameters chosen, the agent would traverse 20,000 steps whilst collecting data on the sequence of visited states and associated reward. Originally studies were run for 10,000 steps, but through several instances of trial-and-error it was found that 20,000 steps displayed more conclusive performance behavior. It is assumed that the hyperparameter trends analyzed from the Gamma and Alpha Study of 10,000 steps remain equivalent at higher step counts. Details of the study for each algorithm can be seen in table 3.3, below. Although each study was run 40 times, there was no averaging or other operations between these datasets. The nature of

this study resulted in more of a qualitative than quantitative analysis which will be discussed in the next section.

Table 3.3: performance study details

Performance Study Specifications						
	Hyperparameter Details			**Study Details**		
	TD Update		E-greedy			
Algorithm	*Alpha*	*Gamma*	*Epsilon*	*Steps*	*Trials*	*Data Acquired*
Q-Learning	0.8	0.95	0.2	20,000	40	Reward, State
Double Q-Learning	0.9	0.5	0.175	20,000	40	Reward, State
SARSA	0.15	0.5	0.2	20,000	40	Reward, State
Expected SARSA	0.8	0.2	0.125	20,000	40	Reward, State

3.3.2 Analysis and Data Visualization

For analysis of the Gamma and Alpha Studies, the data was posed in two ways, both forming a 2D plot. The first approach looked at the cumulative reward at 10,000 steps as the dependent variable and a single hyperparameter, either α or γ, as the independent variable for a fixed ε value. This resulted in a single level curve for each ε value. A number of these curves, known as learning curves, were superimposed onto a single plot for easier comparison. Some curves were omitted from the plots to reduce noise when there was no loss to the inference of trends. These plots will be referred to as the Alpha or Gamma Plot, accordingly. An example of a Gamma Plot can be seen in figure 3.6, below.

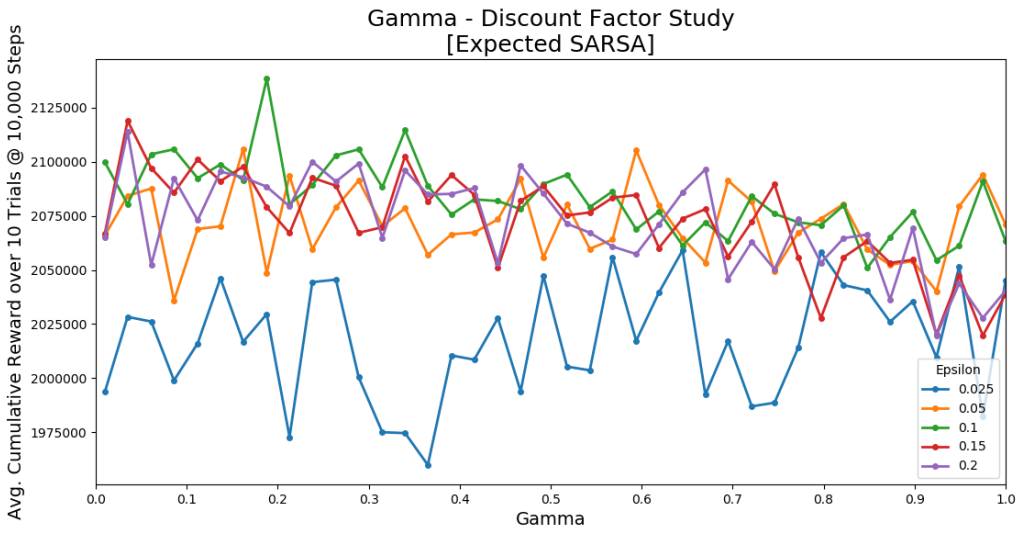

Figure 3.6: example of Gamma Plot from the Expected SARSA hyperparameter study

The second approach produced a similar plot, but with ε as the independent variable and either α or γ at a fixed value. Again, a number of these curves, sufficiently spanning the range of α and γ, were superimposed onto a single plot. This plot will be referred to as the Epsilon-Alpha or Epsilon-Gamma Plot, accordingly. An example of a Epsilon-Alpha Plot can be seen in figure 3.7, below.

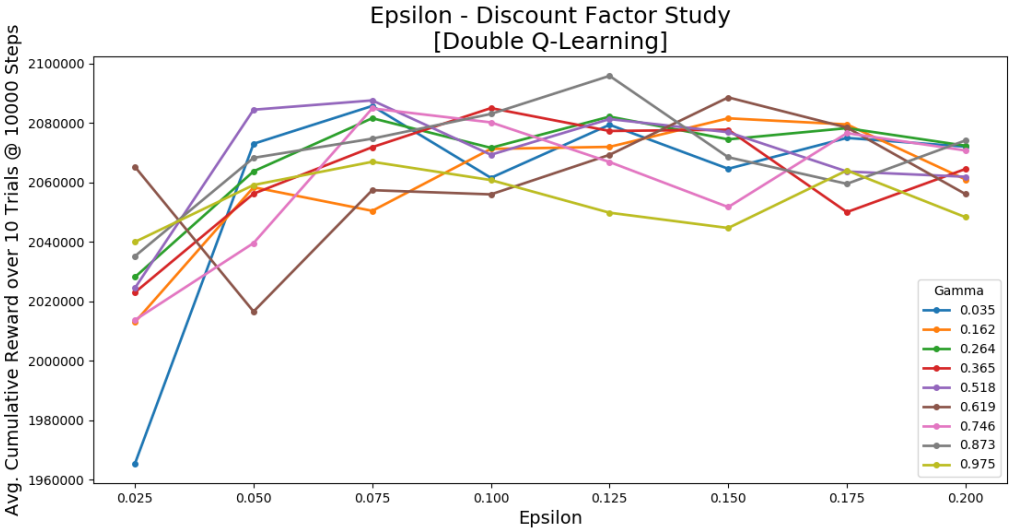

Figure 3.7: example of Epsilon-Alpha Plot from the Double Q-learning hyperparameter study

For the analysis of the algorithmic performance studies, plots of trajectories and instantaneous reward were produced. A trajectory plot consists of some or all the visited states overlying coordinates of the state space. Thus, this visualization is usually only suitable for rectangular and low-dimensional state spaces. To understand the reward of the state space and performance of the trajectory, the reward surface was displayed underneath. This reward surface is a discretized contour plot visualized as a heat map. Two types of trajectory plots were created. The first type, called the Visitation Plot, shows all states visited, with the size of the circular marker proportional to the number of times that state was visited. Also, because the agent is reinitialized randomly when it exists the state space, unique markers were used to show the exited and reinitialized states. For these plots, only one trajectory was plotted to prevent excessive noise and overlap. An example of the a Visitation Plot is shown in figure 3.1, above.

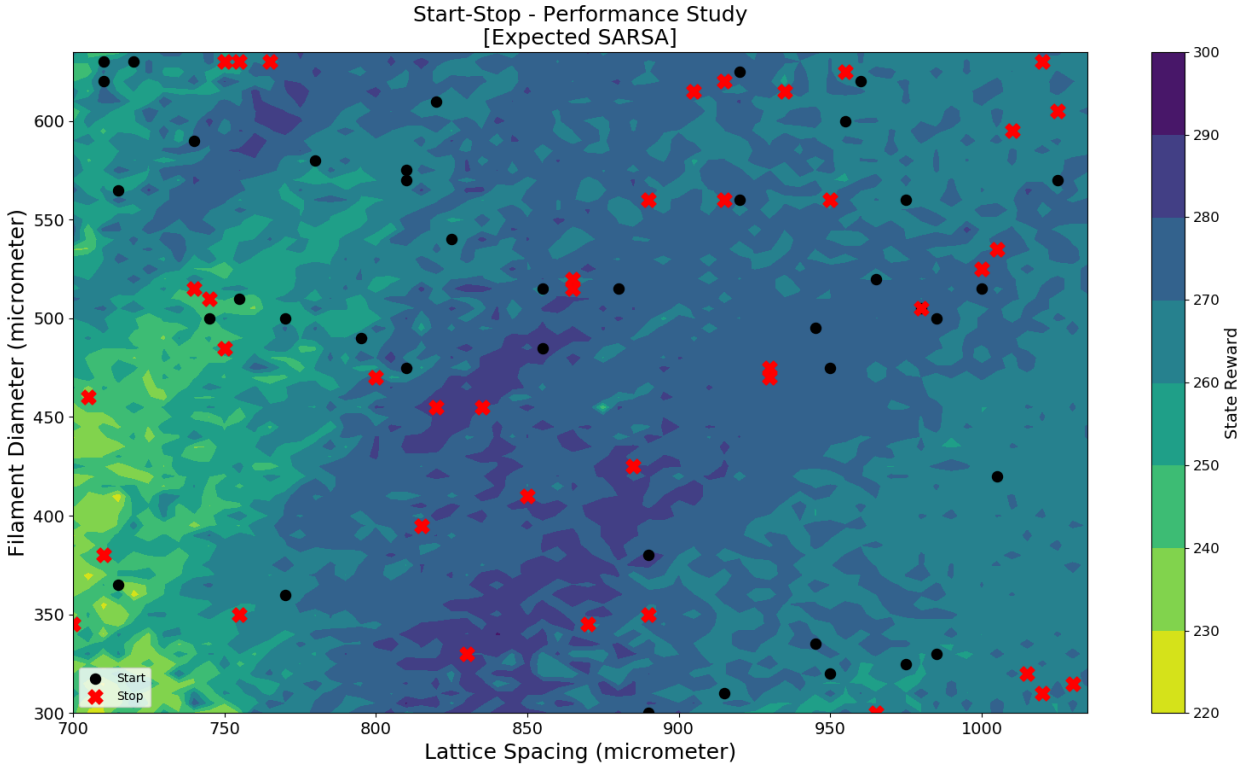

Figure 3.8: example Start-Stop Plot from Expected SARSA algorithm

The other trajectory plot, called the Start-Stop Plot, only displays the first and final state of the full trajectory. In order to see a distribution of final states, 40 trajectories were shown on a single plot. An example of a Start-Stop Plot, can be seen in figure 3.8, above. The final plot, called the Reward Plot, was used to visualize the learning curve. This plot shows the instantaneous reward gathered over the lifespan of the agent's full trajectory. The instances where the agent exists the boundary and receives zero reward has been omitted from these plots to improve clarity and vertical resolution. Both the Visitation and Learning Plots elaborate on instances of data seen in the Start-Stop Plots.

4 Results

4.1 Alpha Study

The data from the two-dimensional hyperparameter study of α and ε, the Alpha and Gamma Study, was compiled into eight plots – two for each algorithm – from which results were interpreted. Each algorithm's notable results will be presented in the following sections. For the sake of conciseness, not all plots are shown below as some trends can be summarized verbally. All excluded plots can be found in the Appendix. Some plots may exclude data of redundant trends to improve clarity of inference.

The first results that will be discussed are from the Alpha Plots that show the cumulative reward at the 10,000th step, averaged over ten trials, against the learning rate, α, for several level curves at fixed ε values. The second results discussed are from the Epsilon-Alpha Plots that show the same data but from a different perspective - the cumulative reward at the 10,000th step, averaged over ten trials, plotted against ε for several level curves at fixed α values. Further explanation of these visuals is above in section 3.3.2.

4.1.1 Results

The first results for the Q-learning algorithm are from the Alpha Plot in figure 4.1, below. It can be seen that there is a noisy distribution, but still an evident linear upward trend in the cumulated reward as α increases to 1. The only outlier is the level curve of ε equal to 0.025, the lowest value. However, there does not seem to be a distinct difference between the cumulative reward for α values less than 0.1.

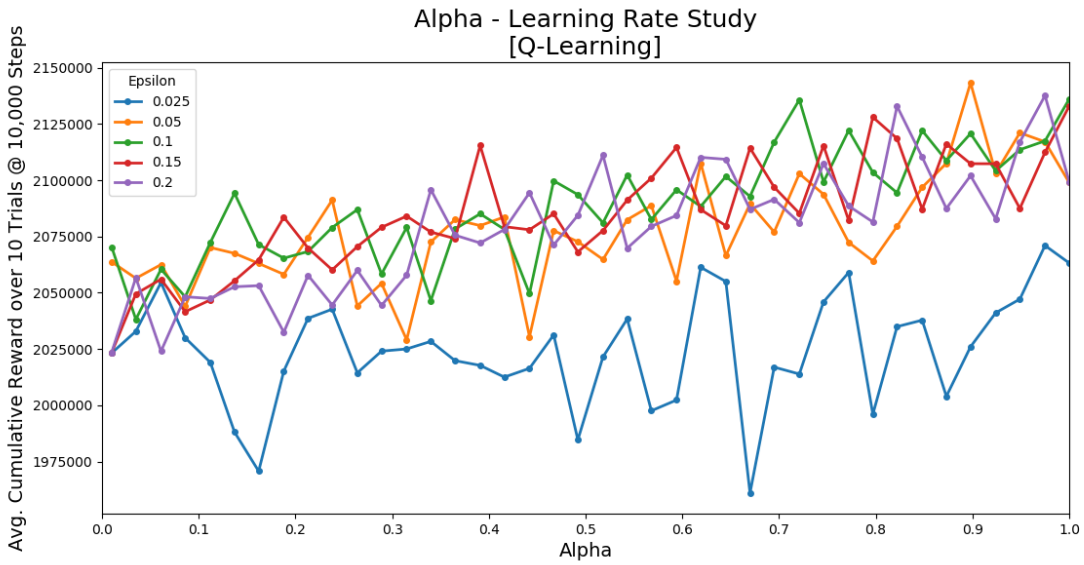

Figure 4.1: Alpha Plot of the Q-learning algorithm for the Alpha Study

The Epsilon-Alpha plot, in figure 4.2, can be seen below. There are two discernible trends: lower α values and the lowest ε value result in low cumulative. Also, cumulative reward with respect to ε, on average, plateaus once ε is greater than 0.075. The only curve that sees a consistent upward trend in cumulative reward after that point is when α is 0.035.

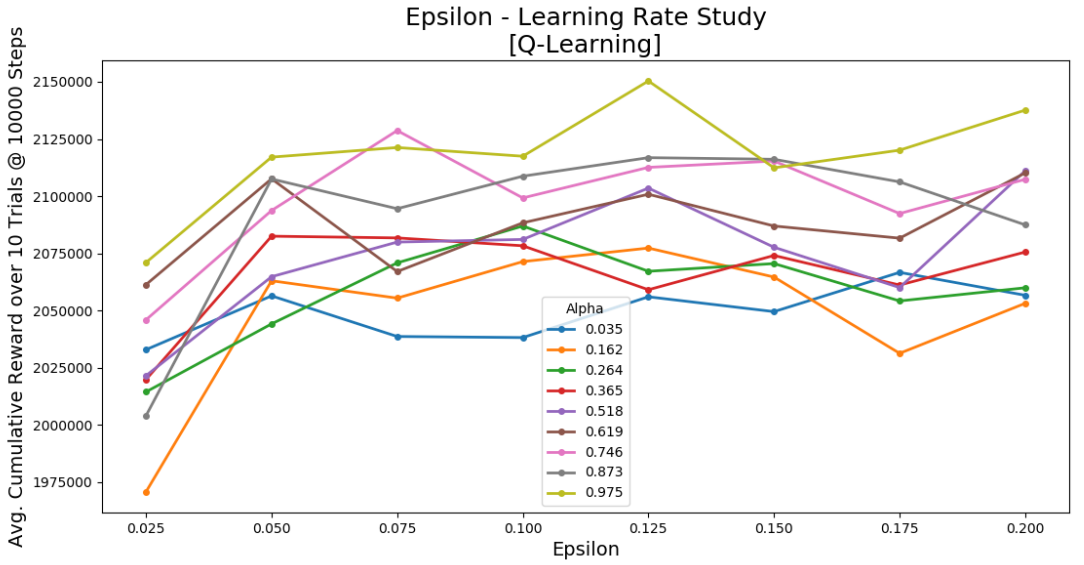

Figure 4.2: Epsilon-Alpha Plot of the Q-learning algorithm for the Alpha Study

The Alpha Plot for double Q-learning, seen in figure B.1 in the appendix, is comparable to the corresponding plot from Q-learning in figure 4.1, above. The two plots have similar distributions of level curves and the same outlier, except the upward trend for Double Q-learning is lesser and begins after α equals 0.03. The results from the Epsilon-Alpha Plot, seen in figure 4.3 below, shows a clear, tight trend – lower ε and α result in lower cumulative reward. Note that the cumulative reward for both plots of Q-learning and Double Q-learning is very comparable.

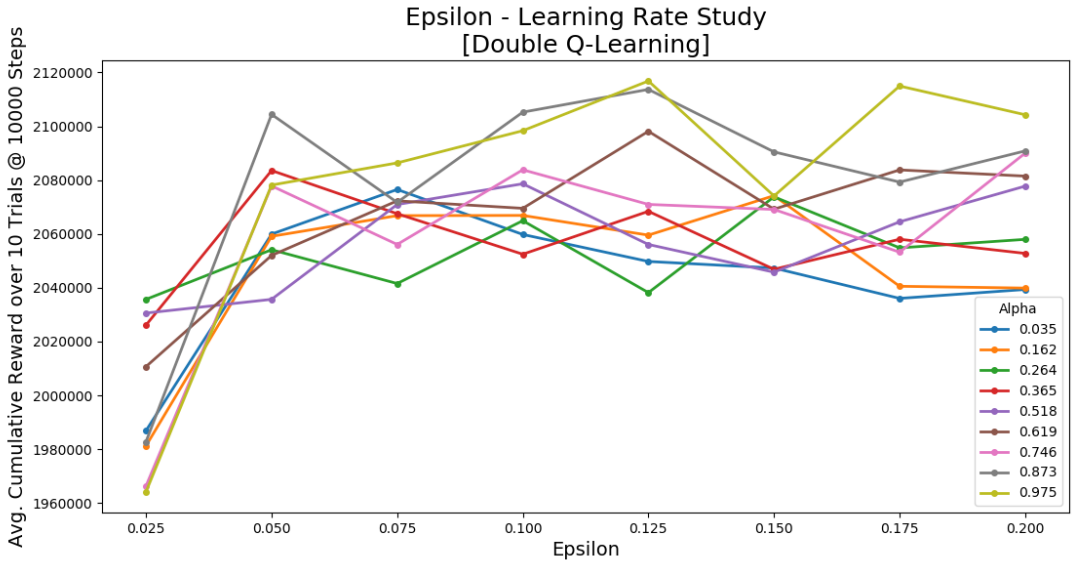

Figure 4.3: Epsilon-Alpha Plot of Double Q-Learning algorithm for Alpha Study

The results from the Alpha Plot for the SARSA algorithm, seen in figure 4.4 below, show a clear downward trend in the cumulative reward as α increases. Again, the ε value equal to 0.025 seems to be a lower outlier when α is equal to 0.5 through 0.9. The range of cumulative reward is comparable to those from Q-learning results. The results from the Epsilon-Alpha Plot, seen in figure B.2 in the Appendix, show similar results found in the Alpha Plot – higher α values result in lower cumulative reward. However, there is not much information to gain on the influence of ε as the curves remain level on average.

Figure 4.4: Alpha Plot of SARSA algorithm for Alpha Study

The Alpha Plot for Expected SARSA, from figure B.3 in the Appendix, shows that ε values of 0.025 and 0.2 clearly results in lower cumulative reward than the rest, with 0.025 being the lowest the majority of the time. The trend of cumulative reward remains relatively level as α increases and the range of cumulative reward values is similar to that of Q-learning. The Epsilon-Alpha Plot, seen in figure 4.5 below, shows a nonlinear trend of cumulative reward as α increases. Only after ε is greater than 0.075 is there a slight downward trend with respect to the learning rate. The epsilon values resulting in maximal cumulative reward lie between 0.05 and 0.1.

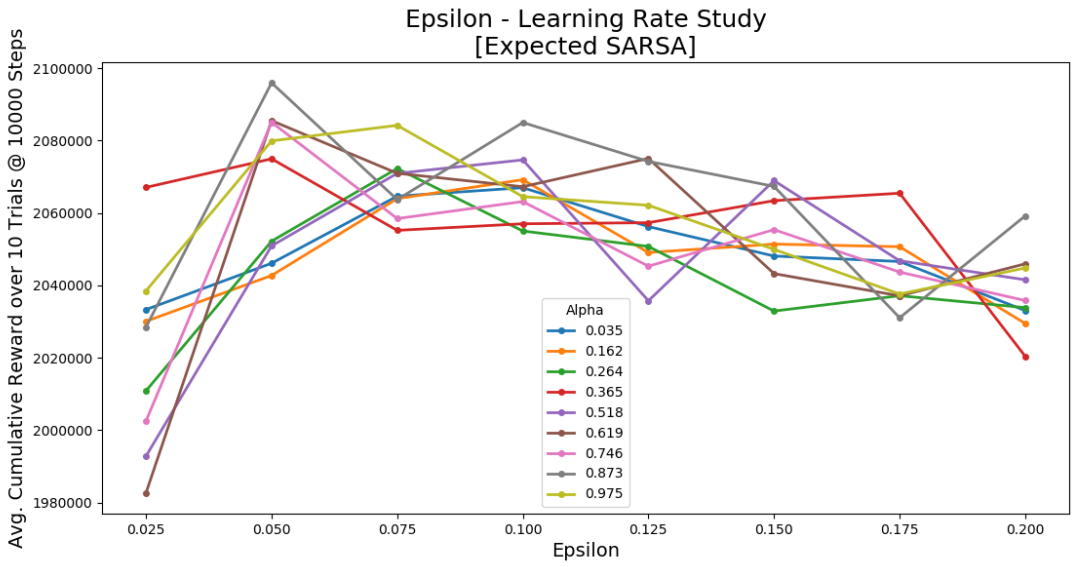

Figure 4.5: Epsilon-Alpha Plot of Expected SARSA algorithm for Alpha Study

4.2 Gamma Study

The data from the two-dimensional Gamma Study of γ and ε was compiled into eight plots, two for each algorithm, from which results were drawn. Each algorithm's notable results will be presented in the following sections. As before, not all plots are shown below, and some trends will be summarized verbally. All excluded plots can be found in the Appendix. Some plots may exclude data of redundant trends to improve clarity.

The first results that will be discussed are from the Gamma Plot that shows the cumulative reward at the 10,000th step, averaged over ten trials, against the discount factor, γ, for several level curves at fixed ε values. The second results discussed are from Epsilon-Gamma plot that shows the same data but from a different perspective - the cumulative reward at the 10,000th step, averaged over ten trials, plotted against ε for several level curves at fixed γ values. Further explanation of these visuals can be found above in section 3.3.2.

4.2.1 Results

Shown below in figure 4.6, the Gamma Plot for Q-learning shows that lower ε values result in lower cumulative reward. The curves of ε equal to 0.025 and 0.05 are notable outliers, with the rest generally remaining between 2,060,000 and 2,120,000 cumulative reward. There is no discernable trend in the cumulative reward as γ increases.

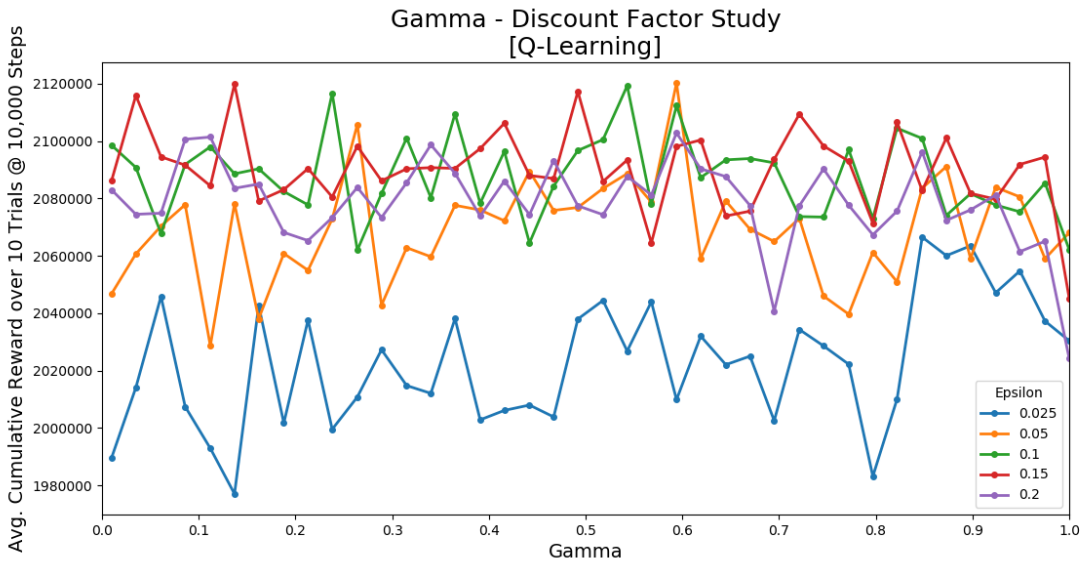

Figure 4.6: Gamma Plot of Q-Learning algorithm for Gamma Study

The Epsilon-Gamma Plot, seen in figure 4.7 below, shows the two trends discovered in the Gamma Plot. However, there is greater clarity in this visualization.

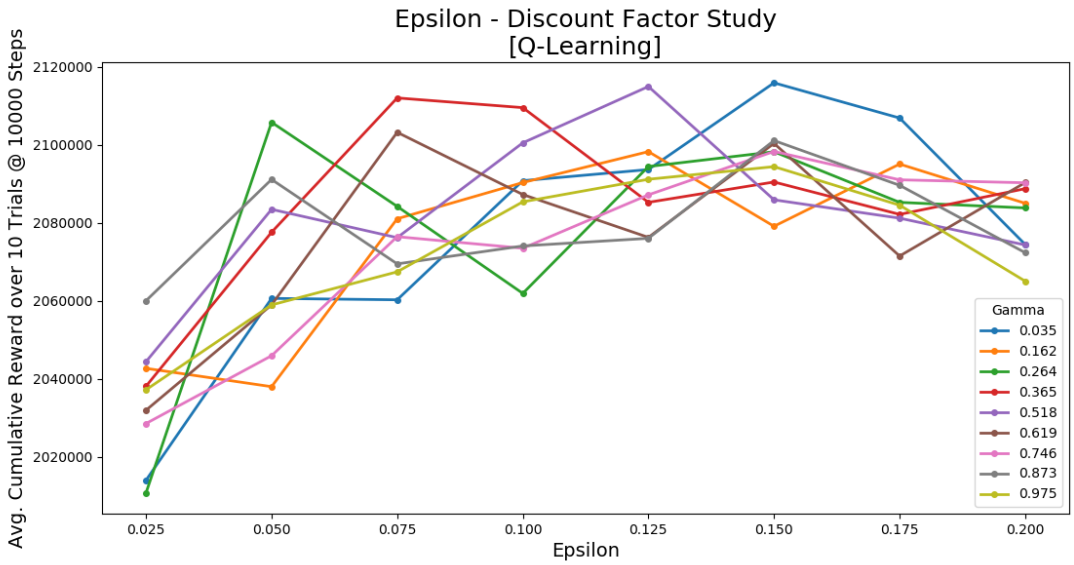

Figure 4.7: Epsilon-Gamma Plot of Q-learning algorithm for Gamma Study

Due to nearly identical trends and for the sake of avoiding redundancy, the Gamma and Epsilon-Gamma Plots for the SARSA and Double Q-learning algorithms can be all summarized from the results of the Q-learning Gamma and Epsilon Gamma Plots seen in figure 4.6 and 4.7, above. Lower ε results in lower cumulative reward and the cumulative reward remains relatively stable across all γ values. However, Expected SARSA's Gamma Plot shows a slight, negatively-sloped trend of the cumulative reward, for all level curves except when ε equals 0.025. This can be seen in figure 4.8, below, and in figure C.3 in the appendix. The remaining plots from can be seen in the appendix.

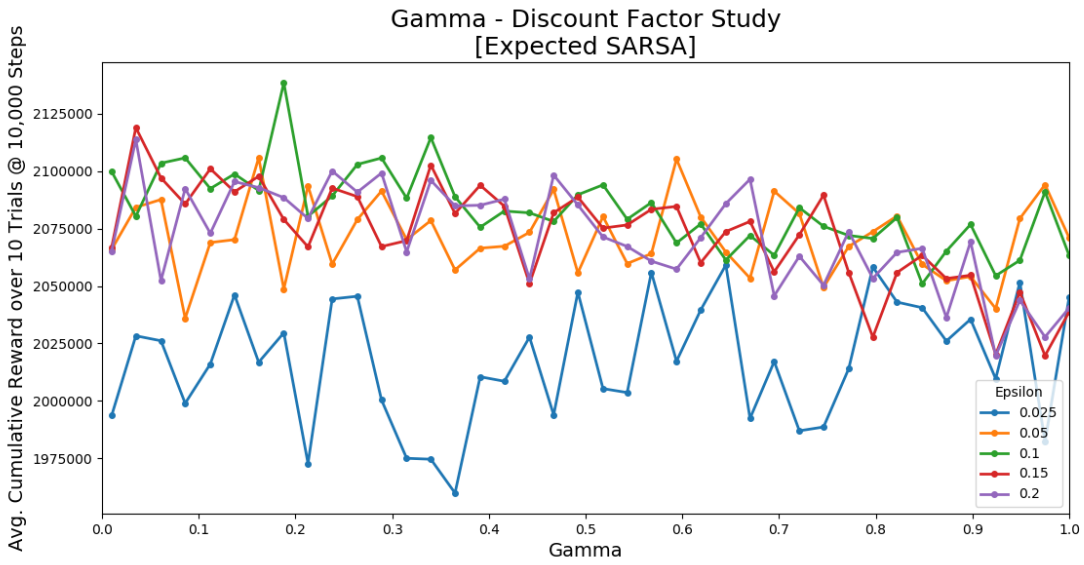

Figure 4.8: Gamma Plot of Expected SARSA algorithm for Gamma Study

4.3 Performance Study

Using the favorable hyperparameters for each algorithm, as outlined in table 3, the data from the performance study was collected and compiled into twelve plots, three for each algorithm, from which results were drawn. Each algorithm's notable results will be presented in the following sections. As before, not all plots are shown in this section and some trends will be summarized verbally. Various excluded plots can be found in the Appendix.

The first results that will be discussed are from the start-stop trajectory – showing the starting and ending state of 40 trajectories for a single algorithm and fixed hyperparameters. The next results will show several visitation trajectories from instances of the start-stop trajectories. Lastly, the instantaneous reward curves will be presented corresponding to the selected visitation trajectories. As there are 40 trajectories for each algorithm, only a few will be selected to represent the results from the trajectories. Again, these three plots are named the Start-Stop Plot, Visitation Plot, and Reward Plot, respectively. Further explanation of these plots is above in section 3.3.2.

4.3.1 Start-Stop Results

The start-stop results for Q-learning can be seen in figure 4.9, below. The starting state locations are randomly distributed around the state space and largely outside the distribution of stopping locations. Only four final states land in the 230-250 reward region. Q-learning was able to completely avoid the lowest reward region. There are six final states are in the highest reward regions of 280 reward and above. The start-stop results for SARSA can be seen in figure 4.10, below. The agent ended much more around the boundary than with Q-learning, but it still managed to end up in the reward region, greater than 280, four times.

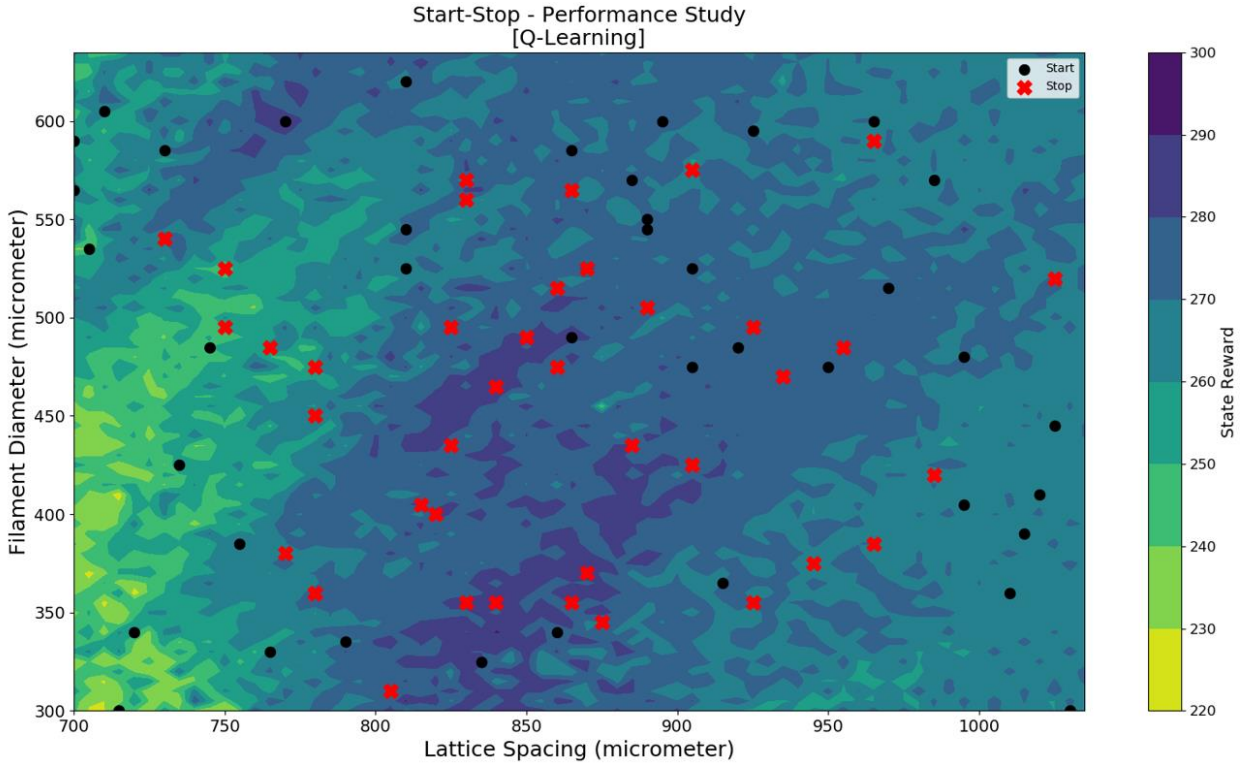

Figure 4.9: Start-Stop Plot of Q-learning algorithm for Performance Study

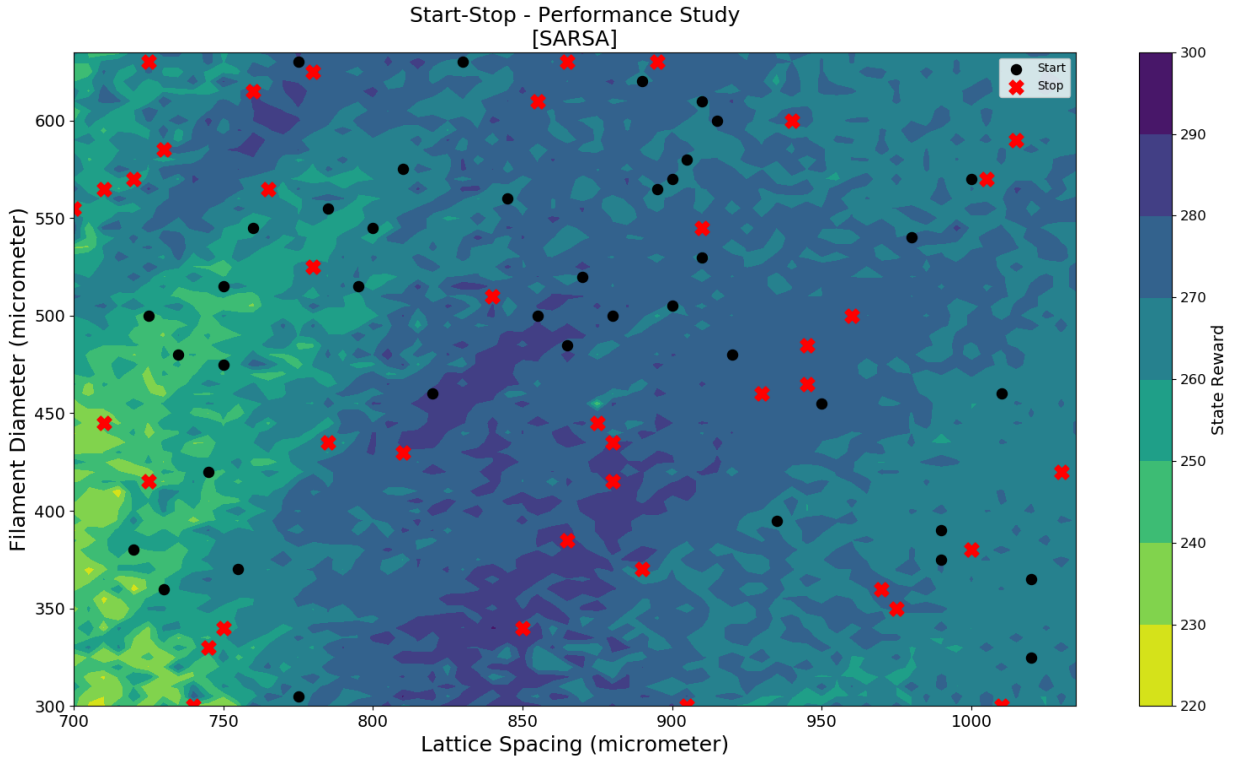

Figure 4.10: Start-Stop Plot of SARSA for Performance Study

The start-stop results for Expected SARSA can be seen in figure 4.11, below. Four to five final states land in the 280 and above reward region. Eight final states landed in the reward region from 220-250. Over a dozen final states are located near the state space borders, unlike for Q-learning. The results for Double Q-learning can be seen in figure 4.12, below. These results of the final states are similar to that of Expected SARSA, with many around the border and a few in the higher reward regions.

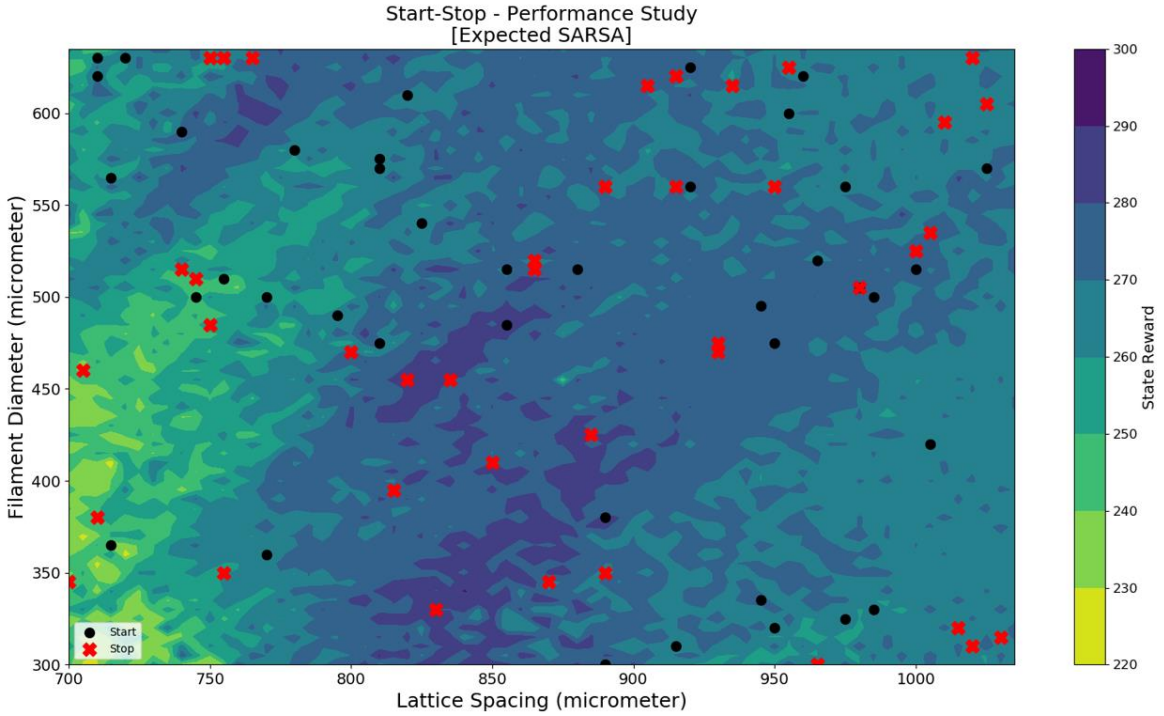

Figure 4.11: Start-Stop Plot of Expected SARSA for Performance Study

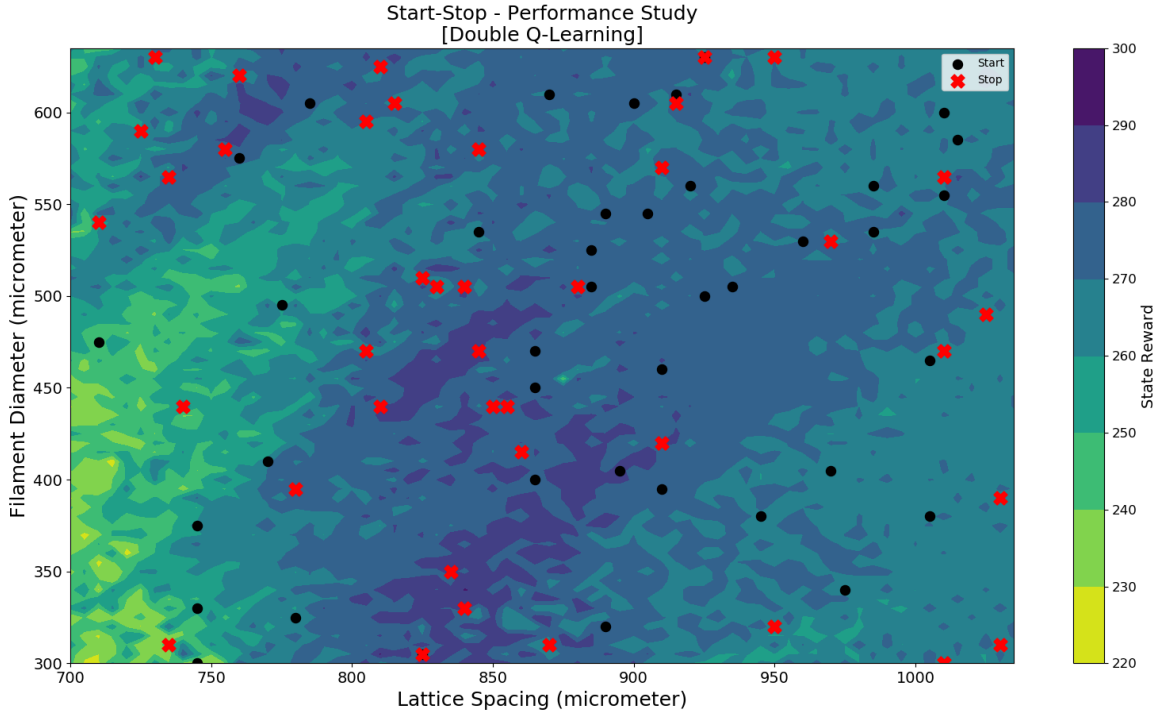

Figure 4.12: Start-Stop Plot of Double Q-Learning for Performance Study

4.3.2 Visitation and Reward Results

The trajectory data from the Start-Stop plots, above, were analyzed further by utilizing the Visitation and Reward Plots. The fact that an agent's path ended in a high or low reward region does not necessarily mean the agent did or did not reap high reward areas. By selecting several trajectories and analyzing their Visitation Plots a better understanding of this uncertainty was gained. Several instances, and their corresponding Reward Plot, are discussed in this section.

A visitation trajectory and its associated instantaneous reward for SARSA can be seen below in figure 4.13 and 4.14, respectively. From both these plots there is no clear indication that the agent located and benefited from high reward regions. At one point, the agent traverses over the highest reward area, but does not remain for many visitations. Although, the most visited states, as indicated by the size of the white marker, are usually in areas surrounded by relatively lower reward regions.

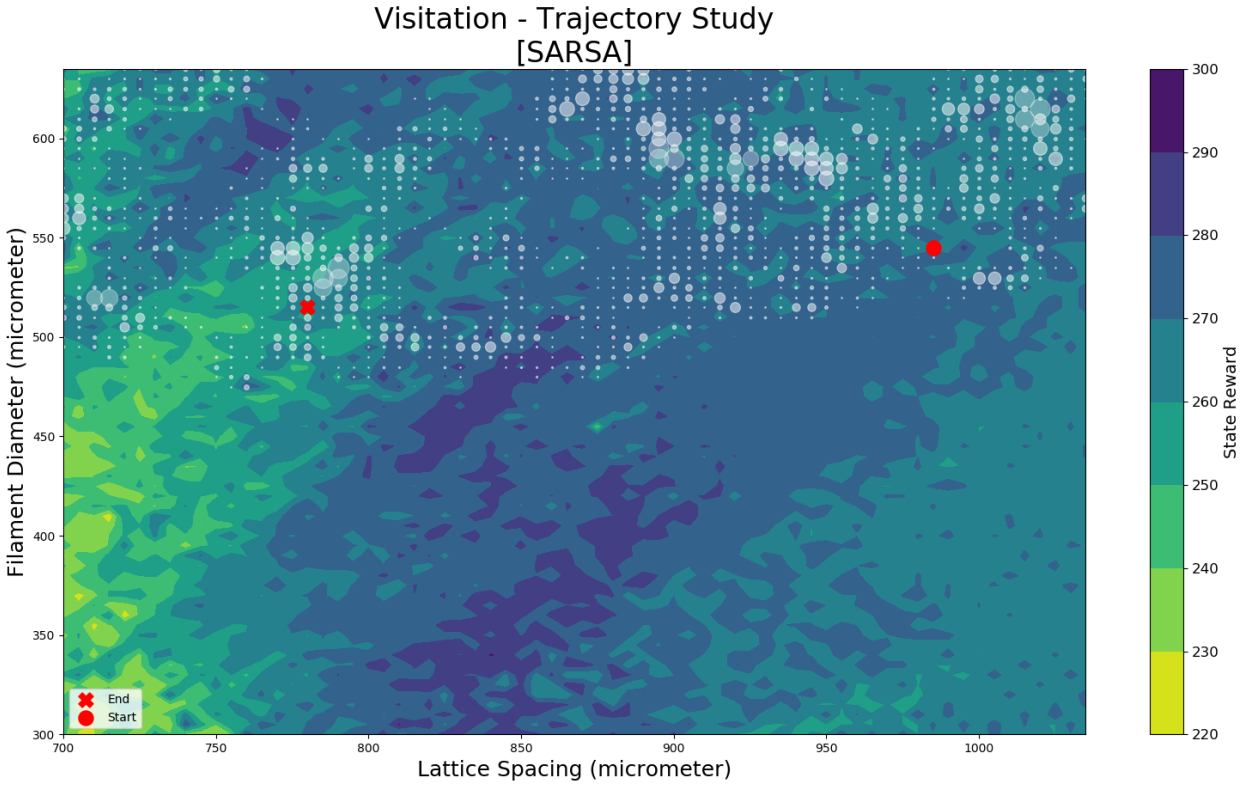

Figure 4.13: instance of Visitation Plot of SARSA algorithm for Performance Study

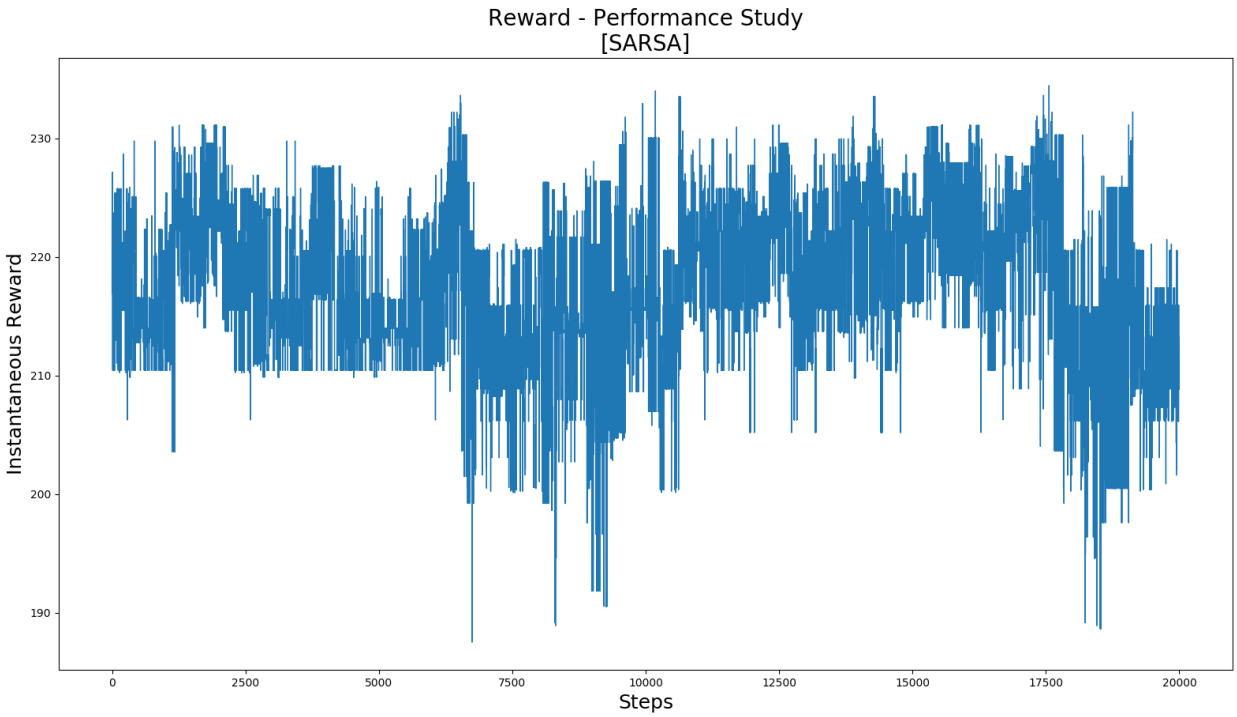

Figure 4.14: instance of Reward Plot of SARSA algorithm for Performance Study

A visitation trajectory and its associated instantaneous reward for Double Q-learning can be seen below in figure 4.15 and 4.16, respectively. Visible from the trajectory, the agent was able to locate the main regions of high reward and remain as more frequent visitations occur in high reward states. Also, the agent managed to end in a high reward state.

Summarizing the rest of the results, including those for Q-learning and Expected SARSA, is necessary for conciseness. The various trajectories among a single algorithm vary greatly and there seems to be no outstanding differences among the four algorithms. Sometimes the agent manages to find the high reward, sticking around for a while or not at all, and other times the agent finds itself stuck at the edges or in lower reward regions for the rest of its entire lifespan. A recurring observation was that the agent managed to identify the gradients of the reward surface and visit the relatively greater reward regions more. Not much extra data or trends were extracted from the Reward Plots. Several more example plots can be seen in Appendix D.

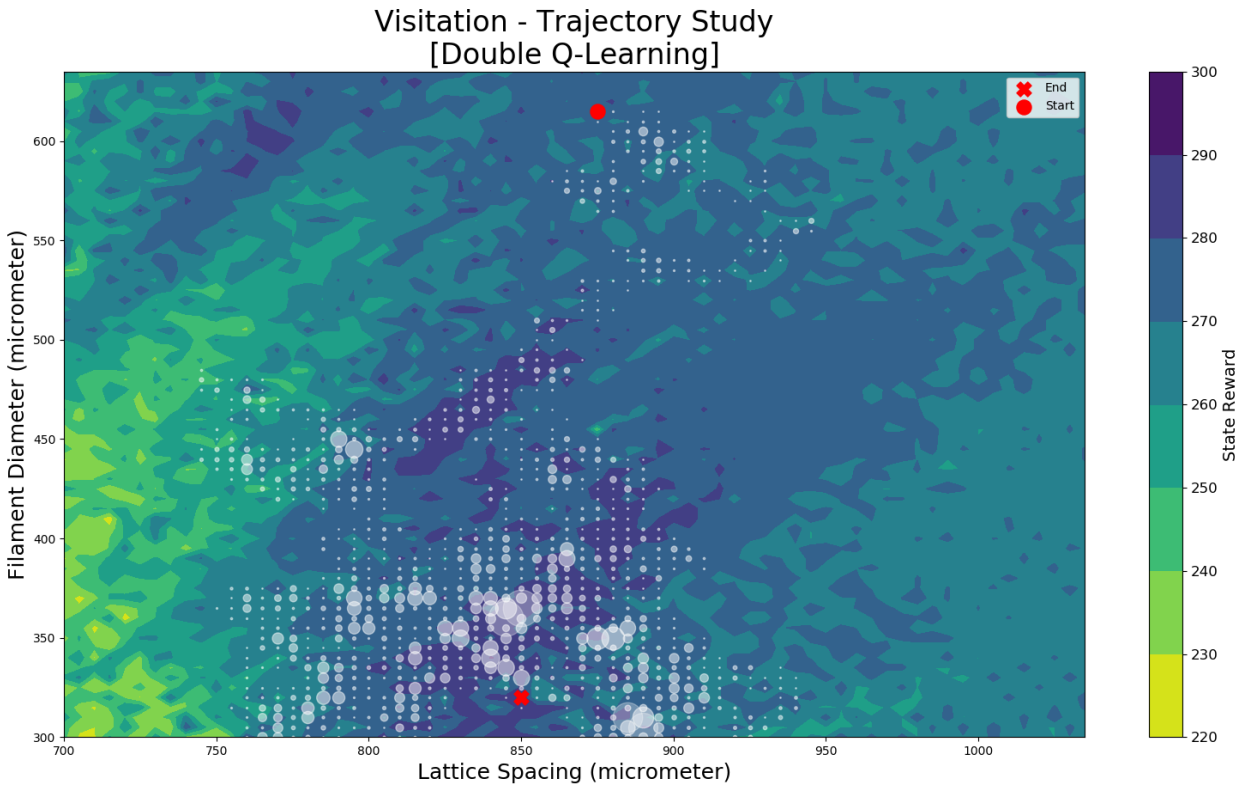

Figure 4.15: instance of Visitation Plot of Double Q-Learning algorithm for Performance Study

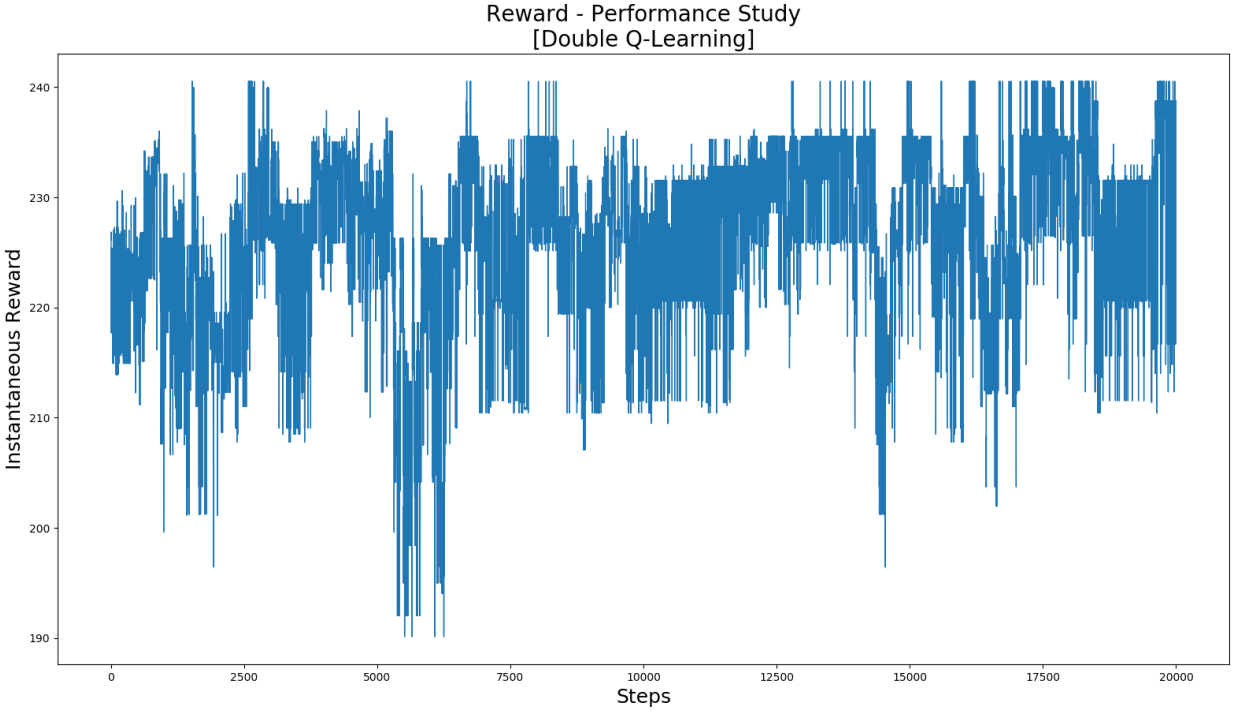

Figure 4.16: instance of Reward Plot of Double Q-Learning algorithm for Performance Study

5 Discussion

5.1 Hyperparameter Study

From analysis of the hyperparameter studies, favorable values for α, γ, and ε were identified. Since an extensive 3-dimensional study was not performed, there is no assertion of optimality, only which configuration of values performed better out of those that were studied. Beginning with the TD update, lower α values were favorable for SARSA, while higher values were favorable for Q-learning. It is not exactly understood why these relationships are opposite, but it is suspected to be due to the fact that SARSA is on-policy and Q-learning is off-policy. Since Q-learning's target policy is greedy, it seems the performance benefits from updating the action-value function more drastically using the maximal action-values. On the other hand, SARSA's target policy incorporates stochasticity from ε-greedy and drastic action-value updates may not always be representative of the optimal policy, therefore, smaller learning rates are preferred to balance the variance. Expected SARSA and Double Q-learning were not sensitive to α. For γ, it appears only Expected SARSA's performances was sensitive to its range of values, while α was fixed at 0.5. Beside Expected SARSA, it is evident that valuing short-term or long-term reward more does not impede target evaluation and TD updates. The deterministic reward function is likely a contributor to this.

Now looking at ε from ε-greedy action-selection: the highest ε value, 0.2, was most universally advantageous, benefiting the performance of Q-learning, SARSA, and Double Q-learning. As for Expected SARSA, there are some differences in trends between the Gamma and Alpha Studies. This discrepancy is due to the fact that there is hardly any overlap in the α and γ configurations - therefore this is likely not a discrepancy at all. Again, determining the better ε value would best be done by a 3-dimensional study. Despite this, it is still noteworthy that ε trends hardly differ between the Alpha and Gamma Studies. This highlights the importance of balancing exploration with exploitation. Lastly, we cannot say that ε is the optimal value because all values were not experimented with, and it likely is not optimal overall because it lies at the upper bound of values tested.

5.2 Performance Study

The data from the start-stop, visitation, and reward plots did a poor job at quantifying how well the algorithms faired in the testbed environment. The Start-Stop Plots did show that the agent was able to find and end on high reward regions for some trials, however, with only a distribution of 40 trajectories it was challenging to discern the probability of whether the algorithm was more likely to succeed or fail. Since the distribution of final states seemed somewhat randomly dispersed among the range of reward regions, we can say there is no evidence these algorithm work consistently in this environment.

Analyzing the Visitation Plots gave further insight into the agent's action behavior and an idea of what the Q-table may look like. On some occasions the agent behaved as one would hope, randomly exploring the state space until finding a high reward zone and remaining around there. Other times the agent behaved in seemingly unusual ways, getting itself stuck in the lowest reward zone or at the boundaries. Without utilizing a method to quantify the 160 Visitation Plots, again, it was challenging to discern the success rates of any algorithm. As stated before in the Results section, there were many examples of the agent traversing along the more beneficial side of the contour lines on the reward surface. This behavior is not unexpected, but worth pointing out because it shows how the agent is able to successfully estimate a sort-of-gradient of action-value at those delimitations. Finally, not much can be said about inferences from the Reward Plots. The noisy and discretized nature of the reward surface resulted in a likewise, and unhelpful, plot of instantaneous reward.

6 Conclusion

Through various hyperparameter and trajectory studies, a better understanding of the four algorithms and their behavior on an environment such as this testbed was achieved. Most evidently, the differences between on-policy and off-policy algorithms were realized from analyzing the learning rate, α. Also, values of ε from 0.15 to 0.2, the highest of the range tested, where concluded to achieve the best balance of exploration and exploitation, thus maximizing

cumulative reward. From the trajectory studies, performance was challenging to conclude, but several instances of promising trajectories occurred for each algorithm.

Reflecting on the completion of this thesis, there were a number of valuable lessons learned. First, more quantitative and conclusive results could have likely been drawn if advanced data analytics methods would have been used to handle the noise and variance. Whether this is through various statistical or better visualization methods, there is likely a way to quantify algorithmic performance more decisively. Or at the least, statistical evaluation could be used to determine how valid the results likely are. Second, it is now understood how helpful it is to visualize an agent's trajectory, first, for debugging purposes. If this trajectory visualization were done in the beginning, the agent's rightward action bias would have been realized for the hyperparameter studies. Lastly, the time and computing power required for extensive computational studies should not be underestimated. Durations and workloads for the studies were estimated but proved to be less than accurate as they depend on various uncontrollable factors. Thus, more than enough time than expected should be allotted to this phase of research. On this same note, proper study automation, data collection, and data management is key to avoid wasted computing effort and time.

In the future, the topics of this thesis could be extended in several ways. With regards to the utilized simulated environment, a stochastic reward function would be more useful to represent the physical system and, thus, would be able to identify favorable hyperparameters and evaluate performance more realistically. Also, with additional time, more trials or a 3-dimensional study could be done to achieve greater certainty and less stochastic results. Other algorithms and action-selection methods could also be explored with a focus on efficient convergence. Convergence rates were not a concern for this thesis, but it is a necessary focus as it is detrimental to the practicality of applying this on a real manufacturing system. Accordingly, promising algorithms could be tested and analyzed on the physical testbed.

References

1. Xu, Xiaochi, Chaitanya Krishna Prasad Vallabh, Zachary James Cleland, and Cetin Cetinkaya. "Phononic Crystal Artifacts for Real-Time In Situ Quality Monitoring in Additive Manufacturing." *JOURNAL OF MANUFACTURING SCIENCE AND ENGINEERING-TRANSACTIONS OF THE ASME* 139, no. 9 (September 2017). https://doi.org/10.1115/1.4036908.
2. Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second edition. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2018.
3. Seijen, H. van, H. van Hasselt, S. Whiteson, and M. Wiering. "A Theoretical and Empirical Analysis of Expected Sarsa." *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL '09. IEEE Symposium On*, March 1, 2009, 177–84. https://doi.org/10.1109/ADPRL.2009.4927542.
4. Szepesvári, Csaba. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, #9. Sand Rafael, Calif.: Morgan & Claypool, 2010.

A Appendix

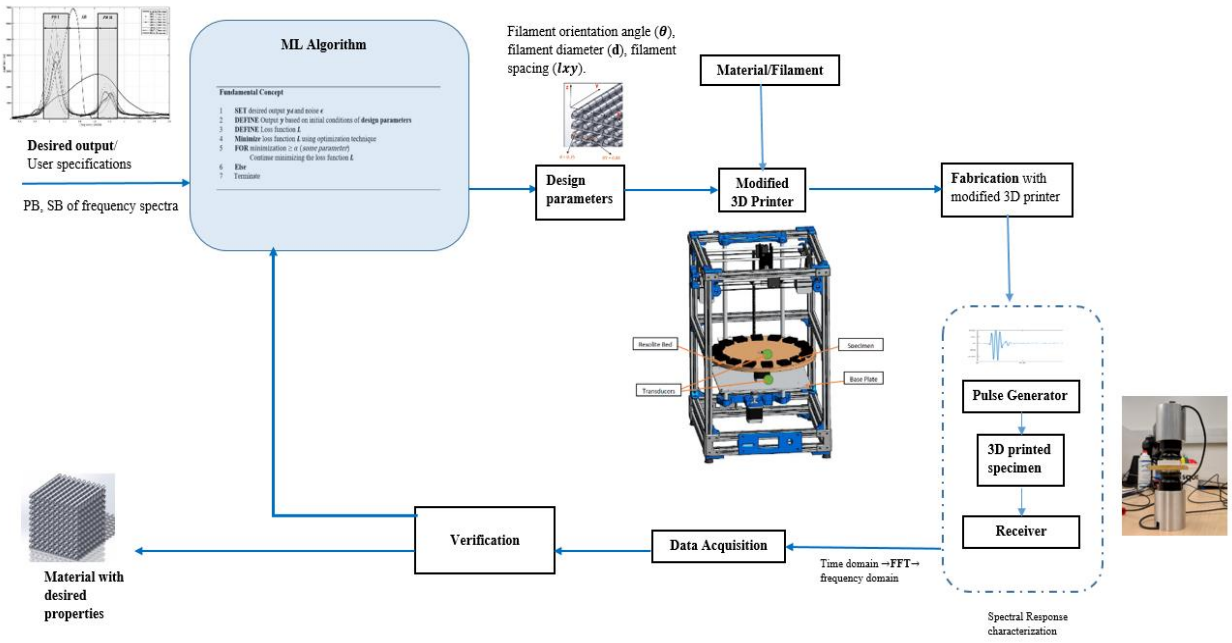

Figure A.1: Physical testbed's process diagram of autonomous manufacturing of phononic crystal

B Appendix – Alpha Study

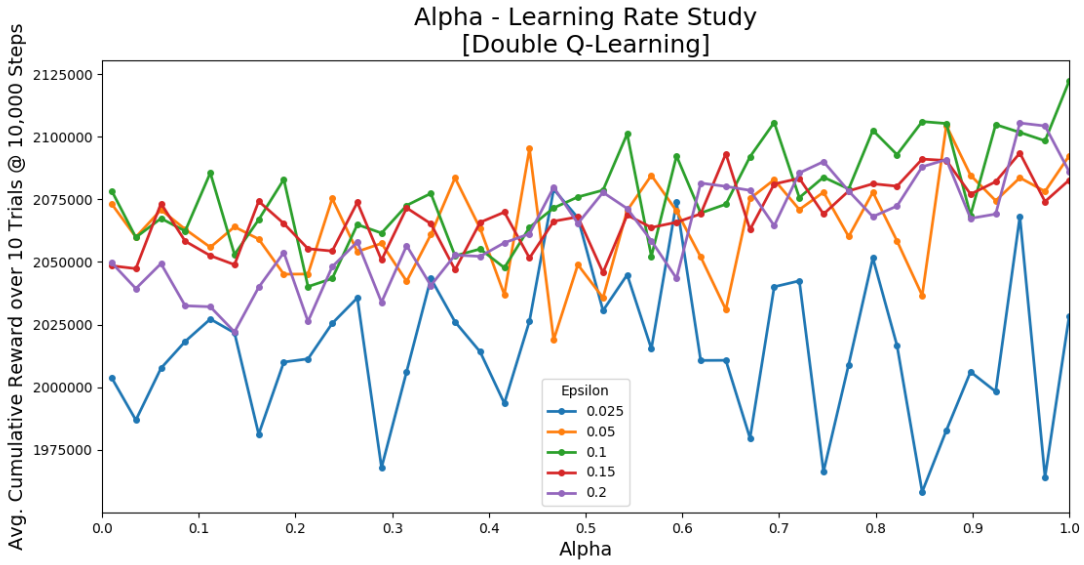

Figure B.1: Alpha Plot of Double Q-learning algorithm for Alpha Study

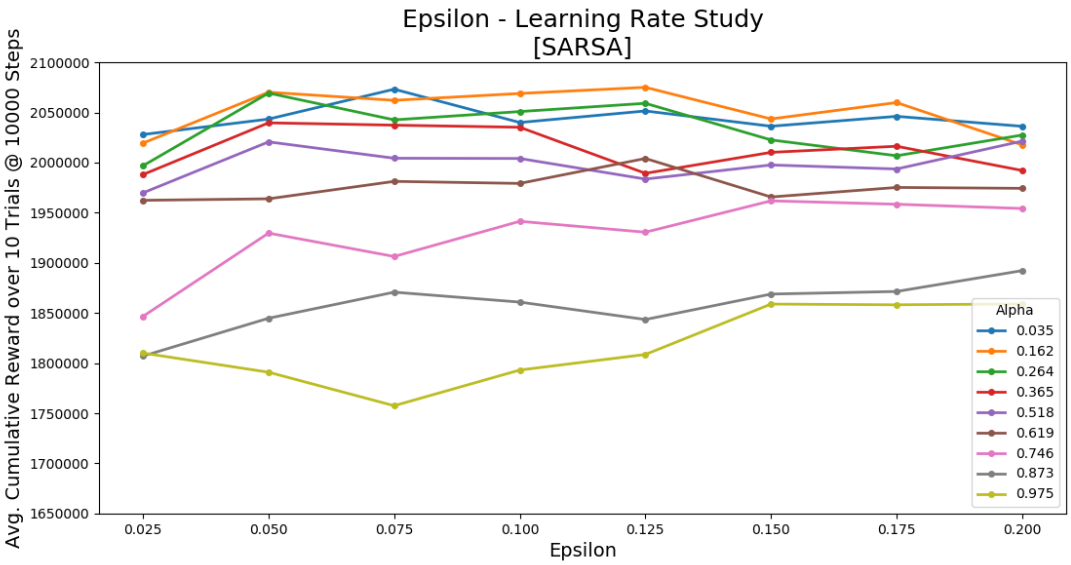

Figure B.2: Epsilon-Alpha Plot of SARSA algorithm for Alpha Study

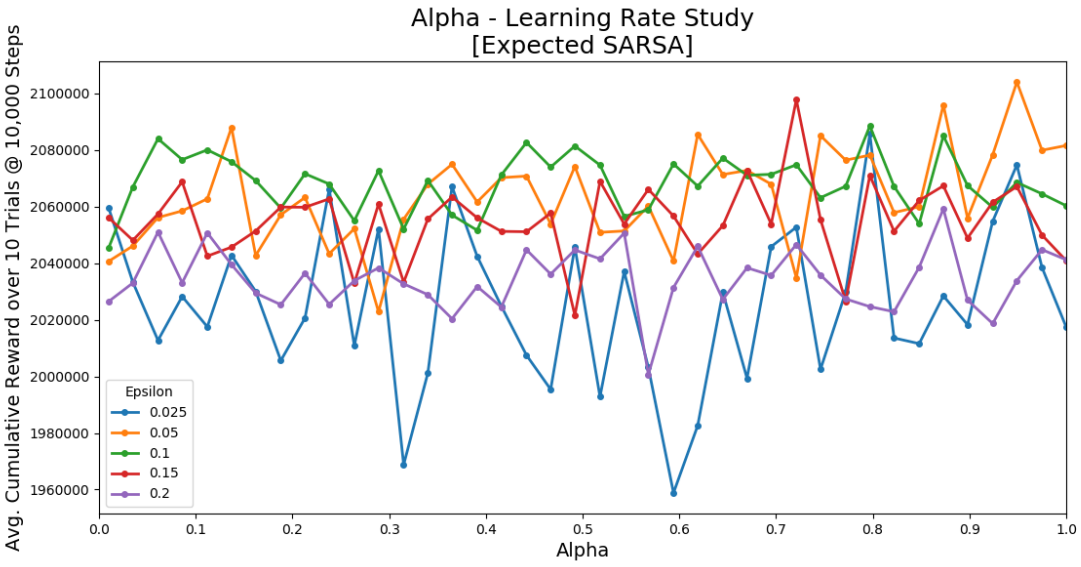

Figure B.3: Alpha Plot of Expected SARSA algorithm for Alpha Study

C Appendix – Gamma Study

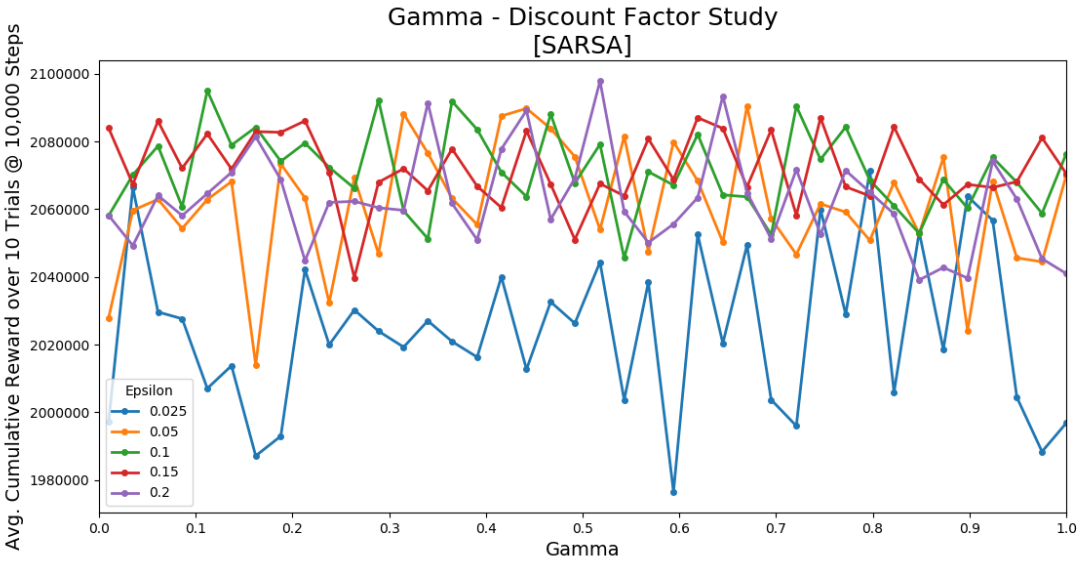

Figure C.1: Gamma Plot of SARSA algorithm for Gamma Study

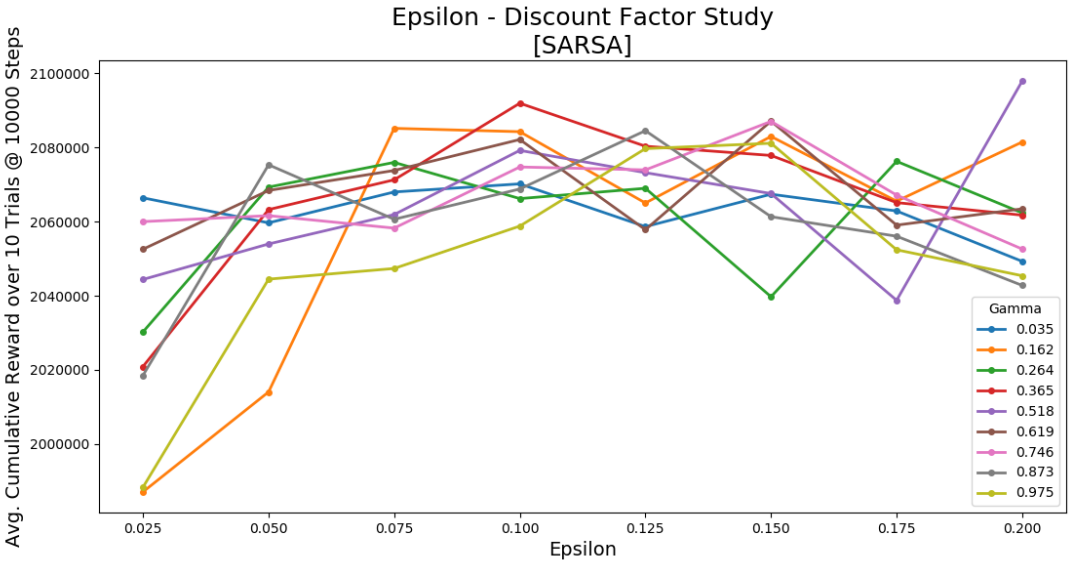

Figure C.2: Epsilon-Gamma Plot of SARSA algorithm for Gamma Study

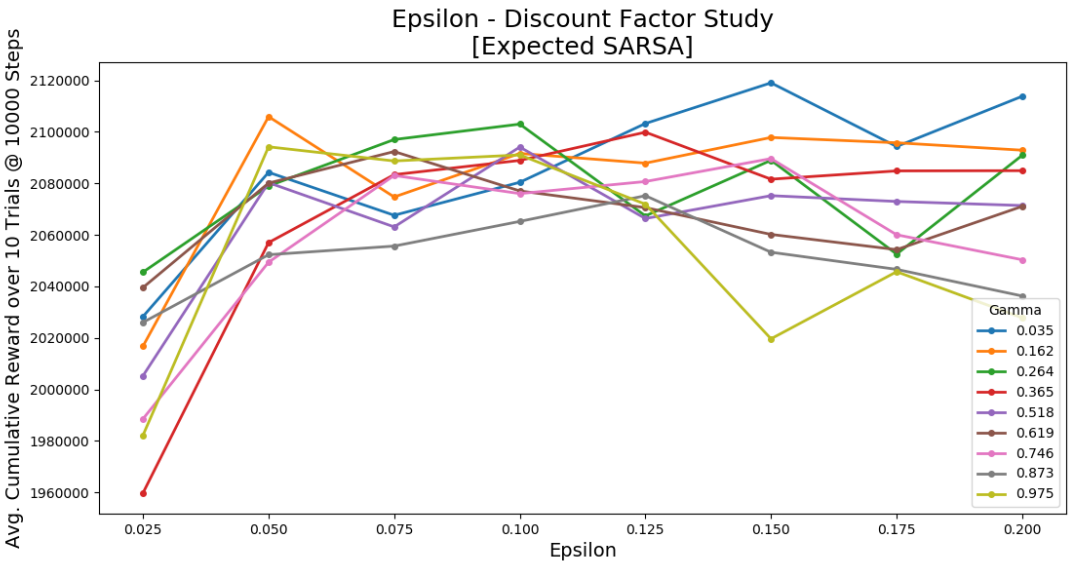

Figure C.3: Epsilon-Gamma Plot of Expected SARSA algorithm for Gamma Study

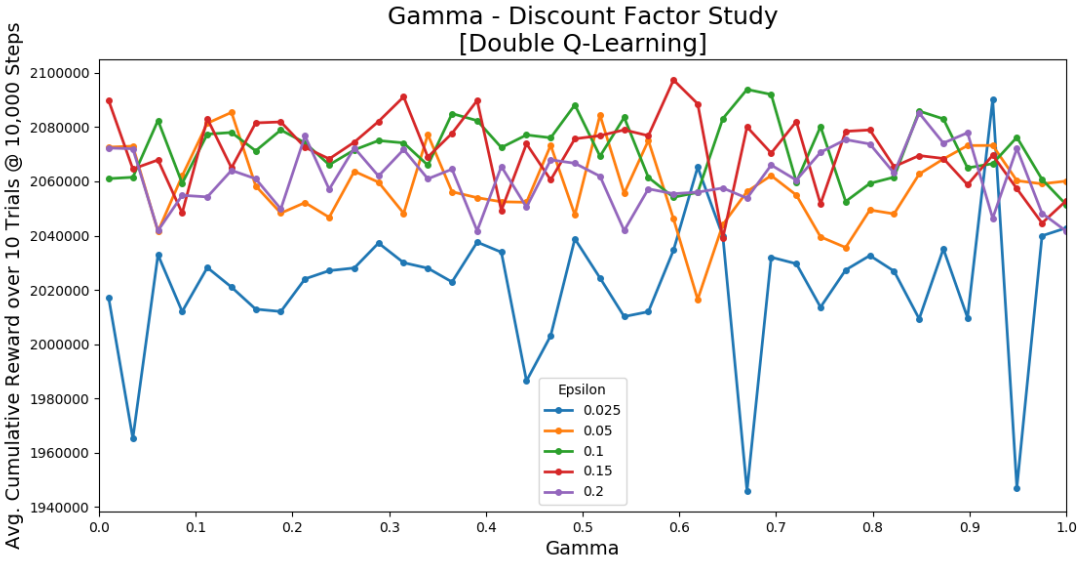

Figure C.4: Gamma Plot of Double Q-learning algorithm for Gamma Study

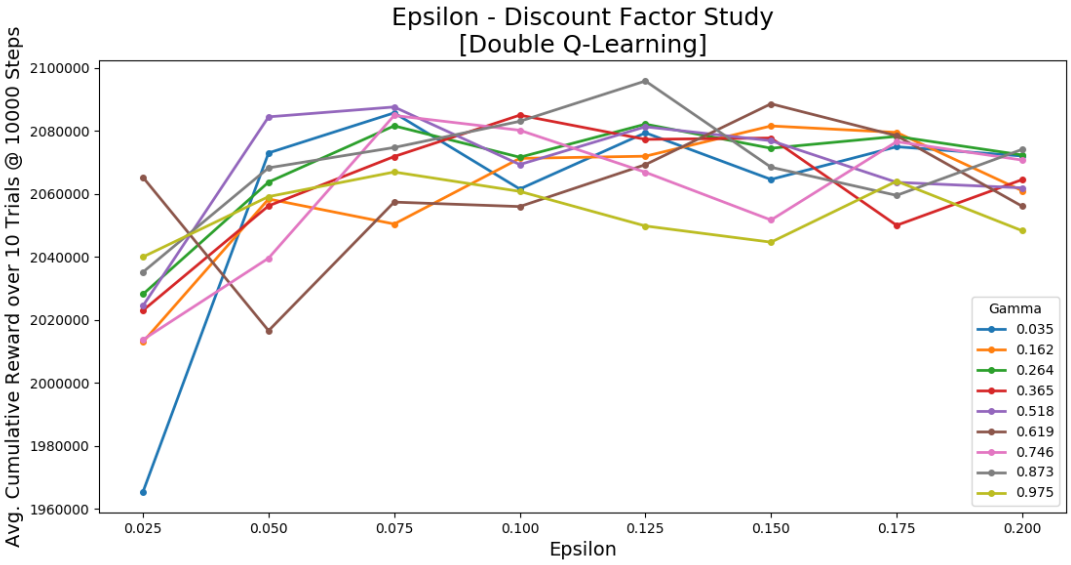

Figure C.5: Epsilon-Gamma Plot of Double Q-learning algorithm for Gamma Study

D Appendix – Performance Study

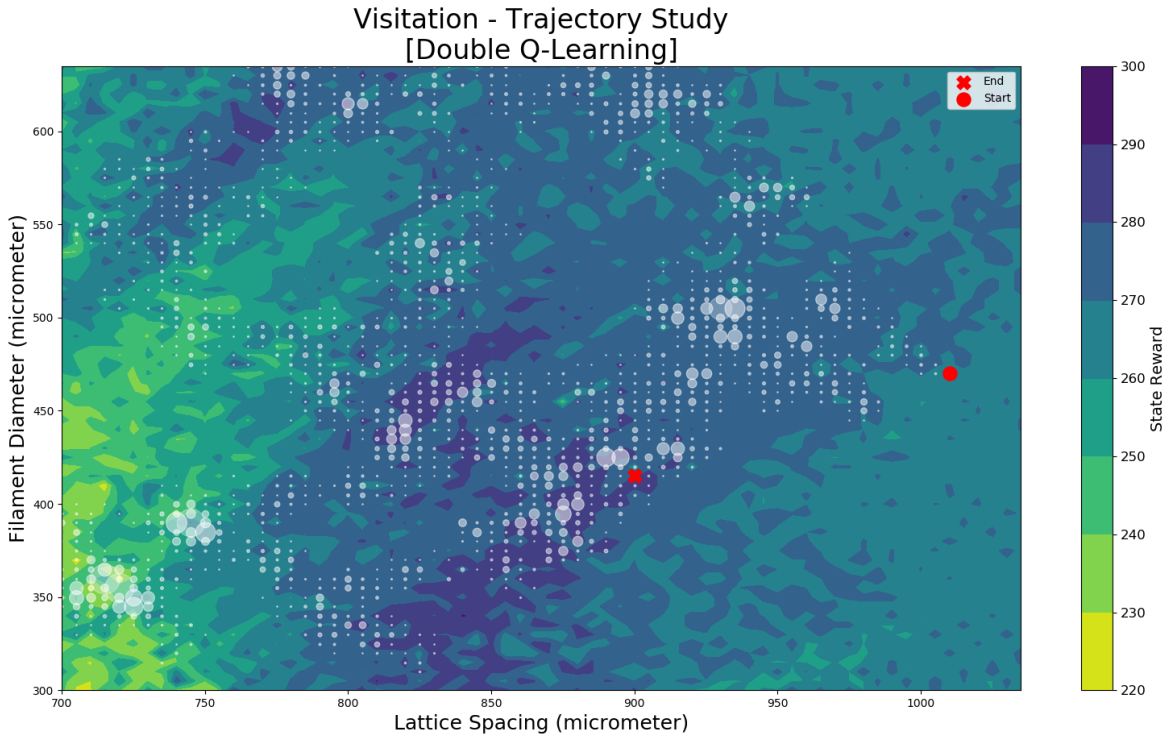

Figure D.1: (20) Visitation Plot of Double Q-Learning for Performance Study

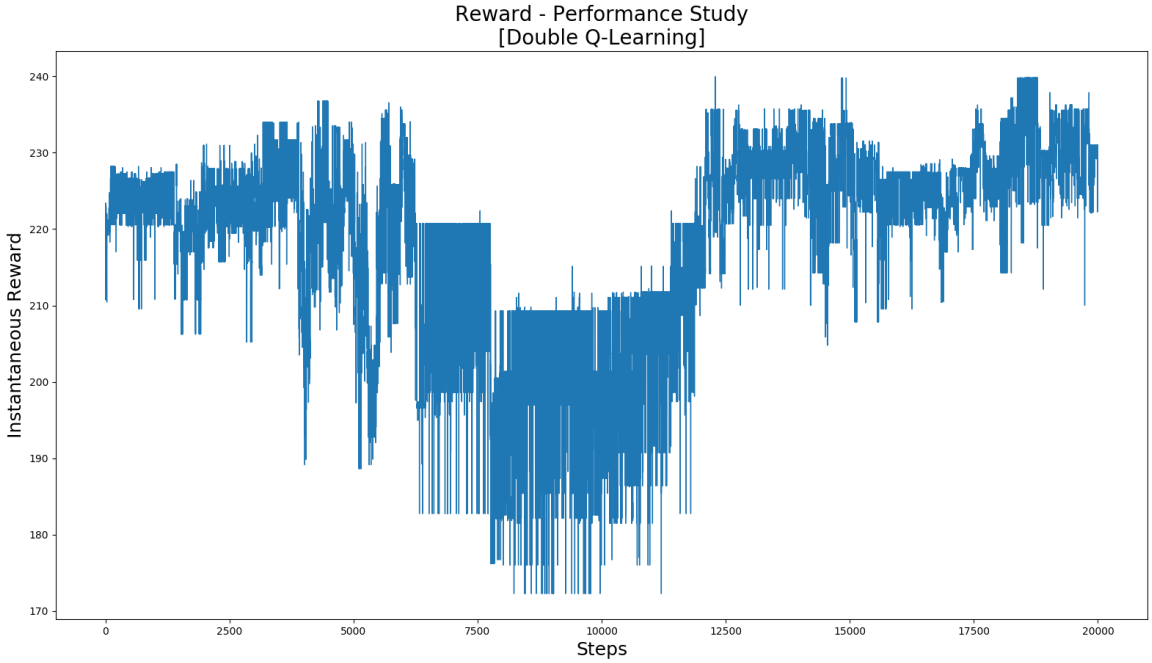

Figure D.2: (20) Reward Plot of Double Q-Learning for Performance Study

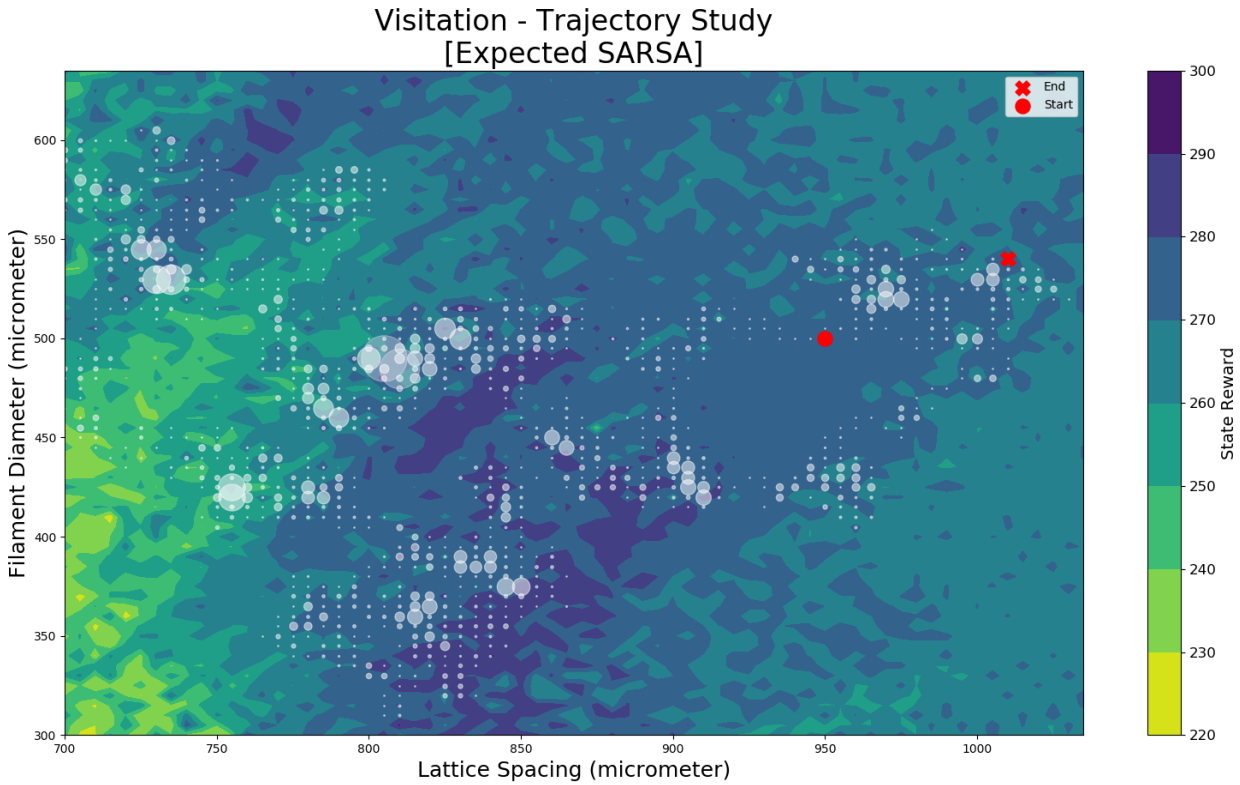

Figure D.3: Visitation Plot of Expected SARSA for Performance Study

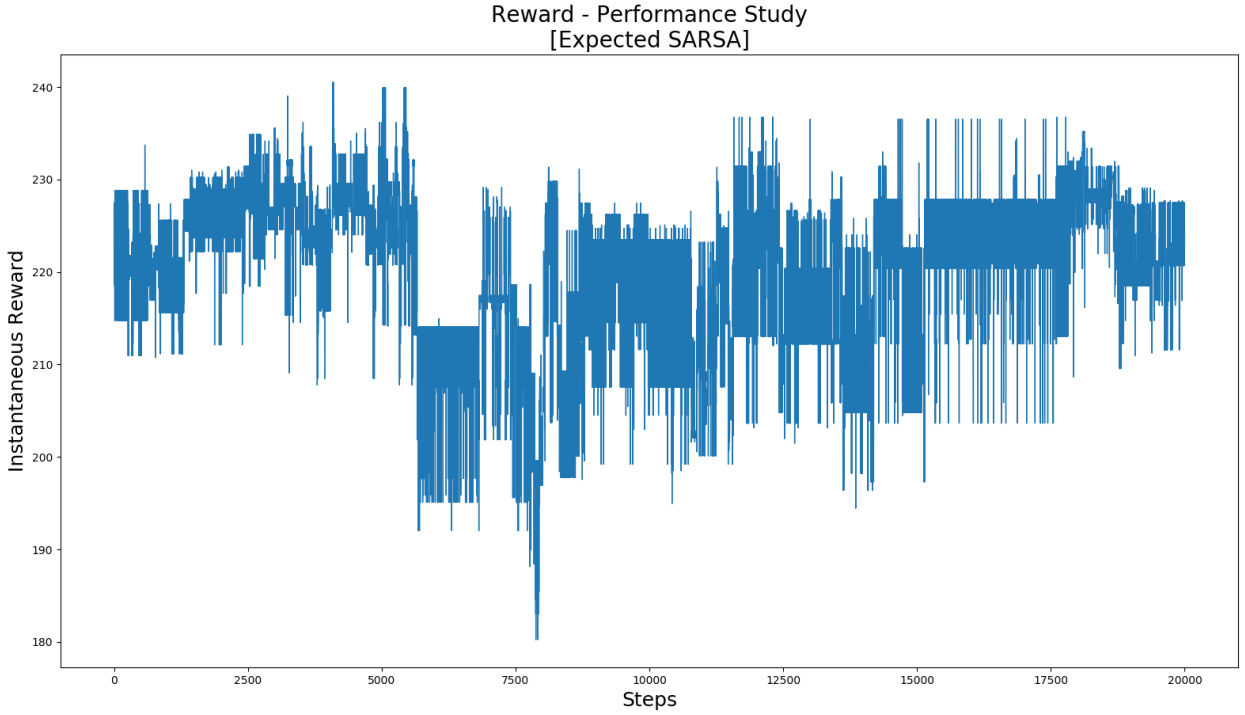

Figure D.4: Reward Plot of Expected SARSA for Performance Study